Syntactica

Syntactica

Version 1.0 (for Java)

by
Richard K. Larson
David S. Warren
Juliana Freire de Lima e Silva
Konstantinos Sagonas

S.V. Ramanan, ***Java Port***

Supported in part by the National Science Foundation

The MIT Press
Cambridge, Massachusetts
London, England

©2010 Massachusetts Institute of Technology

All rights reserved. No part of this book may be reproduced in any form by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher.

This format is intended to reduce the cost of publishing certain works in book form and to shorten the gap between editorial preparation and final publication. Detailed editing and composition have been avoided by distributing the text of this book directly from the authors' prepared documents.

This project was supported in part by the National Science Foundation. Opinions expressed are those of the authors and not necessarily those of the Foundation.

Contents

Acknowledgments 11

CHAPTER 1 *W*elcome to Syntactica 1

What is Syntactica? 1
The Grammar as Science Project 2
How Can Syntactica Be Used? 2
Conventions Employed in This Manual 3

CHAPTER 2 *I*nstalling Syntactica 5

System Requirements 5
Installation Procedure for Microsoft Windows® 5
Installation Procedure for Apple Mac OS X® 5

CHAPTER 3 *S*yntactica Basics 7

The Two Core Functions of Syntactica 7
The General Layout of Syntactica 10
Syntactica Windows and Files 11
Opening Files 11
Creating Files 13
Editing Windows and Files 16
Saving Files 16

Loading Files 16
Sample Session: Building Phrase Markers 19
Sample Session: Transforming Phrase markers 25

CHAPTER 4 Working with Rules and Rule Files 31
The Rule Window 31
Entering Rules.....33
Adding, Editing, Saving and Printing Rules 34
Sorting Rules.....37
Including Rules From Other Rule Files 37
Setting and Changing Heads 39
Setting Rule Defaults 40

CHAPTER 5 Working with Lexicons and Lexicon Files 43
The Lexicon Window.....43
Entering Lexical Information.....45
Adding, Editing, Saving and Printing Lexicons 50
Sorting a Lexicon.....52
Including Lexical Items From Other Lexicon Files.....52
More on Features 54
How Syntactica Distinguishes Lexical Items.....56

CHAPTER 6 TreeViewer..... 61
The TreeViewer Window 61

Loading Rules and Lexicons 63
Inputting Sentences & Generating Phrase markers 65
Viewing Alternative Structures 68
Changing Tree Display Format 69
Displaying Triangles in Trees 71
Printing Trees 71
Saving Trees as EPS Images 72
The PF Display 72
Sending Sentences and Trees 73

CHAPTER 7 Sentence Files, Tree Files, and Grammars 75
Sentence Files 75
Tree Files 81
Grammars 87

CHAPTER 8 Transformations 91
The Transforms Panel 91
Types of Operations 92
Deletion 93
Indexing 93
Copying 95
Left-Adjunction 97
Right-Adjunction 100
Substitution 101
Canceling and Undoing Operations 104

Setting the Empty Symbol & Trace Symbol.....105

CHAPTER 9 Constraints on Transformations..... 107

General Constraints on Movement Operations.....107

Constraints on Substitutions 113

Constraints on Copying.....117

CHAPTER 10 Heads, Features, and Complements 121

Feature and Rule Interactions 121

Complement, Feature and Rule Interactions.....123

Common Errors in Specifying Complements.....126

CHAPTER 11 Head-Final Languages..... 129

Entering Head-Final Rules.....129

Creating a Head-Final Lexicon.....131

Index 135

Acknowledgments

We are grateful to all those involved with the original Syntactica for NeXTSTEP application, and to the National Science Foundation for funding its development under NSF grant USE-915041.

We thank Amy Brand and Ada Brunstein at MIT Press for supporting the Grammar as Science project at all stages. Thanks also to Hiroko Yamakido for initial editorial assistance with the Syntactica for JAVA manual, and to Jessica Bechard for much-appreciated final bug-testing.

RKL: I wish to express my particular indebtedness to S. V. Ramanan, for taking on the job of porting Syntactica to Java, for executing the port with care, impressive speed and unfailing good humor, for suggesting and implementing many improvements to the original NeXTSTEP application, and for making himself available for bug fixes and rebuilds all the way across the world in Chennai, India. I am profoundly grateful.

CHAPTER 1

Welcome to Syntactica

What is Syntactica?

Syntactica is a software application tool designed to let you study natural language structure in a fun, interactive way. The program provides a simple interface for creating grammars, for viewing the structures that they assign to natural language expressions, and for transforming those structures by syntactic operations.

In Syntactica, a grammar consists of a set of context-free phrase structure rules and (typically) a lexicon. The application provides a window interface for creating sets of phrase structure rules, and for creating sets of lexical items (lexicons).

Rules and lexicons are loaded into the TreeViewer window where they are used to generate phrase-markers (or tree diagrams). The user enters a sentence or other expression and Syntactica tries to generate a phrase-marker for it using the rules and lexicon that have been loaded. When more than one structure is available, Syntactica displays the range.

Multiple rule and lexicon windows can be open at any one time, making it easy to load alternate grammars, and to test and compare their results. Phrase-markers can be saved for later viewing and printing, either as files or individual images. Sentence and Tree windows allow you to conveniently collect the sentences and trees generated by a grammar, or to work with an assigned set.

Syntactica permits many aspects of syntactic theory to be explored. The rule and lexicon windows allow you to assign and control the percolation of syntactic features. The TreeViewer window lets you to perform a variety of formal operations on trees by simply pointing, clicking and using the Transforms panel. Syntactica also allows you to control various constraints on operations, including an elementary version of Subjacency.

The Grammar as Science Project

Syntactica was produced as part of the National Science Foundation project *Grammar As Science* (GAS), conducted at the State University of New York at Stony Brook from 1991-1995. This project was a joint venture by the Stony Brook Departments of Linguistics and Computer Science. The leading idea of GAS was, and remains, that linguistics provides a uniquely effective medium for introducing students from a wide variety of academic backgrounds to the principles of scientific reasoning and method.

The GAS emphasis on developing scientific reasoning skills has strongly guided the design of Syntactica. Our chief goal has been to produce an application tool that, while lifting some of the calculative burden from the student, leaves fundamental decisions about how to analyze a given expression squarely in the hands of the user. For example, headedness does not follow automatically from phrase structure configuration in Syntactica, but rather must be explicitly declared. Students learn what headedness amounts to by explicitly manipulating this aspect of structure. Likewise Bounding Nodes for movement are not fixed antecedently as some specific set (IP, NP) but rather are explicitly declared by the user. By being able to vary Bounding Nodes the user can thus explore modern parametric theory in a very direct way.

How Can Syntactica Be Used?

Syntactica is designed for use in introductory syntax courses in linguistics, or in any linguistics or language structure courses with a basic syntax component.

The application can be used as a stand-alone tool for individual self-study, or as a study-aid in a conventional lecture format course. It can also be used as the basis for a "laboratory" component in syntax.

Although designed primarily as a teaching tool for students, Syntactica also has features that will appeal to professionals. The application can be employed as a useful "derivation calculator" in following out complex syntactic derivations involving many steps and operations. Furthermore, Syntactica can be used to create high-quality, camera-ready EPS tree images that can be scaled and edited, and incorporated into professional publications and presentations. (All tree diagrams appearing in this manual were created by Syntactica.)

Conventions Employed in This Manual

This manual employs the following typographic and naming conventions that the user should be aware of.

- Keyboard instructions in this manual are indicated in Courier typeface, and follow normal conventions. For example, an instruction to press the "o" key while holding down the control key would appear as: "`enter control o`".

- Instructions to select a menu or submenu choice are indicated using **boldface** and the character "**>**". For example, the Transforms panel in Syntactica is accessed by clicking on the Options menu item in the Main menu, and selecting the Transforms submenu item within it. An instruction to select the Transforms panel would thus be abbreviated as **Options > Transforms**.

CHAPTER 2

Installing Syntactica

This chapter describes what kind of operating system you must have to run Syntactica, how to install it, and how to register for support.

System Requirements

Syntactica Version 1.0 for Java currently runs under the Microsoft Windows® and Mac OSX®.

Installation Procedure for Microsoft Windows®

To install Syntactica on machines running Microsoft Windows versions 98, ME, XP, Vista, or WIN 7:

1. Download *InstallSynt.EXE* and double click it. The Installer application will come up.
2. Follow directions for installation.

Installation is now complete!

Installation Procedure for Apple Mac OS X®

To install Syntactica on machines running Apple's Mac OSX operating system:

1. Download *Syntactica.pkg.zip* and double click it. An installer package called *Syntactica.pkg* will appear in the same location.
2. Double-click *Syntactica.pkg*. A window entitled "Welcome to the Syntactica Installer" will come up.
3. Follow directions for installation.

Installation is now complete!

USER NOTE: Syntactica sample files will be installed concurrently with Syntactica in the same location. Also installed is XSB, the Prolog engine that performs all computation for Syntactica. XSB extends Prolog with memoization and a more complete handling of negation.

CHAPTER 3

Syntactica Basics

This chapter briefly describes basic notions of Syntactica, including:

- the two core functions of the application,
- the general layout of the application,
- the central operations of opening, editing, saving and loading.

Also included are two brief sample sessions. More detailed information on these topics can be found in the individual chapters of this manual, and in the on-line Help files for Syntactica (located in **Help** in the application menu).

The Two Core Functions of Syntactica

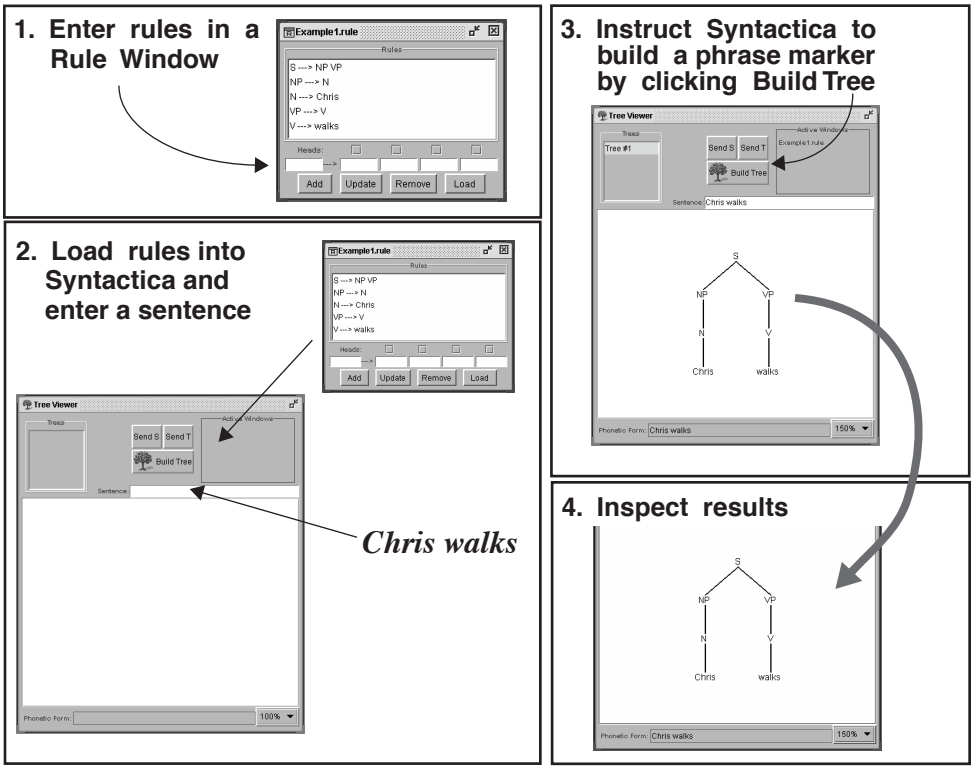
Syntactica is designed to perform two basic operations:

- Generating trees from syntactic rules and lexicons,
- Transforming trees.

The user follows the same general sequence of steps in both cases; he/she:

1. Enters information in Syntactica windows,
2. Loads window information into Syntactica,
3. Instructs Syntactica to perform a computation,
4. Inspects the results.

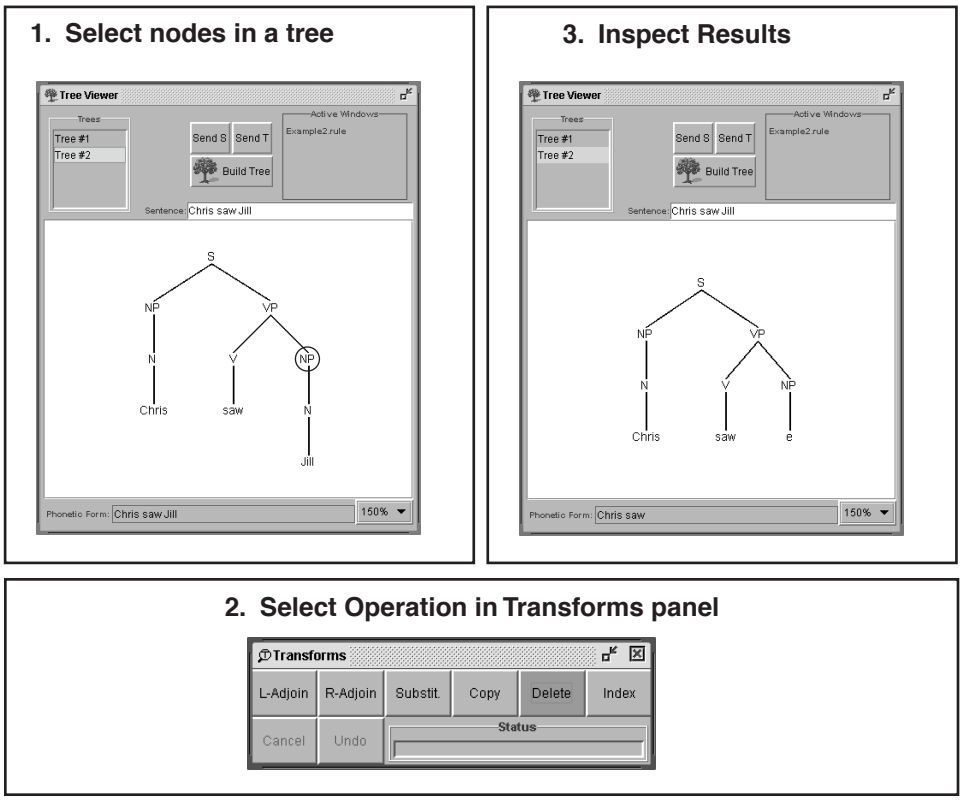
Here is a typical sequence of steps in generating a tree:



GENERATING PHRASE MARKERS IN SYNTACTICA

The user enters phrase structure rules. This information is loaded into Syntactica, together with a sentence. Clicking the Build Tree button instructs Syntactica to build a tree for the sentence using the rules that have been loaded. Syntactica displays the results, which the user then inspects and evaluates.

The same general sequence of steps occurs with transforming a tree. Again, here is a typical session:



TRANSFORMING PHRASE MARKERS IN SYNTACTICA

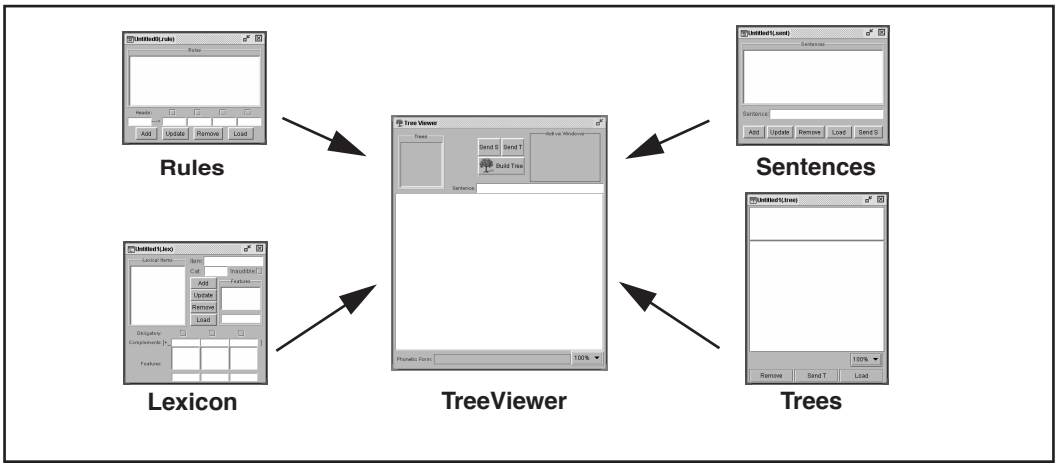
The user works with a tree that has already been built. By clicking on nodes in the tree and a selected operation button in the Transforms panel, the user instructs Syntactica to alter the tree in a specific way. Syntactica displays the transformed tree, which the user can then inspect.

The General Layout of Syntactica

Syntactica contains a number of windows, including:

- a TreeViewer window,
- a Rule window,
- a Lexicon Rule window,
- a Sentence window,
- a Tree window,

The TreeViewer window is central. The other windows can be viewed as arranged around, and communicating with, TreeViewer:



THE RELATIONS AMONG WINDOWS IN SYNTACTICA

Information is entered in the arrowed windows, loaded into TreeViewer, and used to compute results. Results of computations are displayed in the TreeViewer window.

Syntactica Windows and Files

Syntactica windows are used to create and display Syntactica files. For all window types there is a corresponding file type, identified by its own extension:

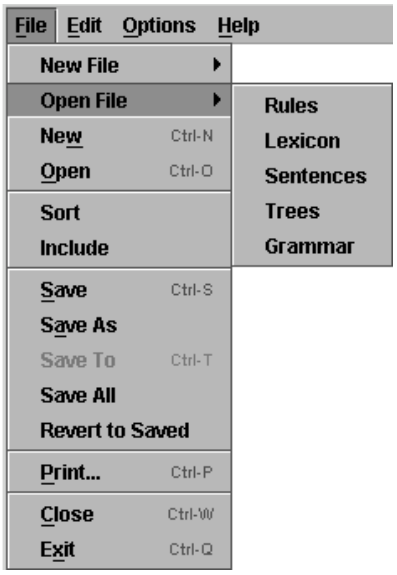
Window Type:	Rule Window	Lexical Window	Sentence Window	Tree Window
File Type:	rule file	lexicon file	sentence file	tree file
File extension:	*.rule*	*.lex*	*.sent*	*.tree*

The same core operations are performed in all Syntactica windows. Those operations are:

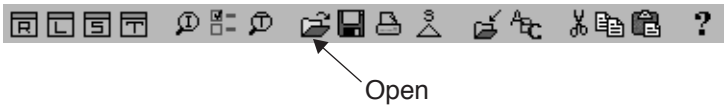
- opening files,
- creating files,
- editing files,
- saving files,
- loading files.

Opening Files

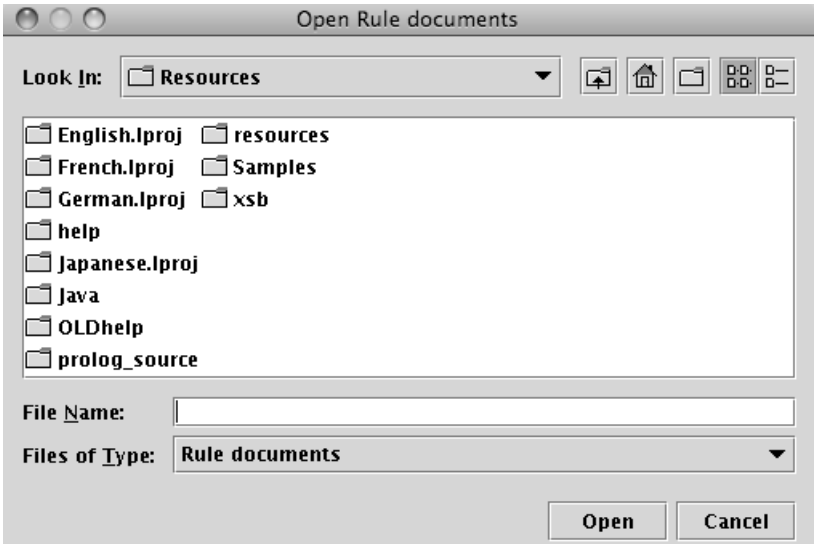
Existing files are opened in Syntactica using the application menu. To open a Syntactica file, choose **File > Open File** in the menu. Select an appropriate file type from the **Open File** submenu. A corresponding Open panel will appear. Files open in the appropriate type of window. An alternative way to open with menus uses the Key window. Whenever a window of a given type is the Key window, choosing **File > Open** from the menu brings up an Open panel of that type:



SHORTCUT! Click the Open icon in the Button bar:



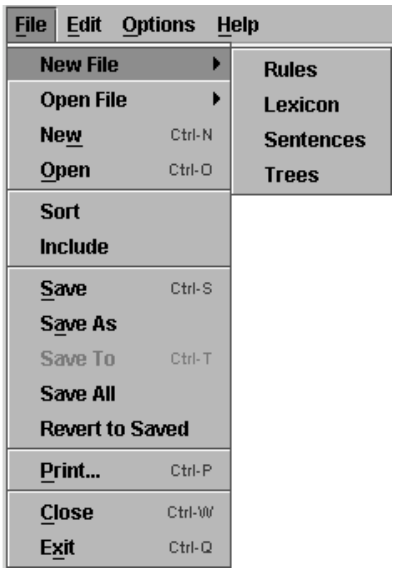
EXAMPLE: Choose **File > Open File > Rules**. An Open Rule documents panel appears. Locate the *Samples* folder that was installed with Syntactica



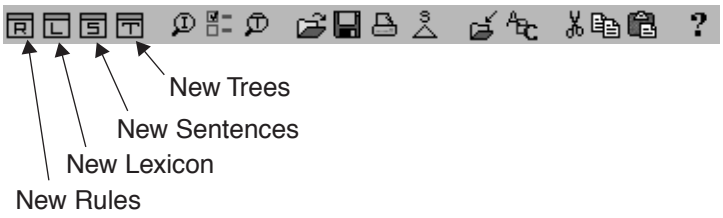
Open *Samples,* either by double-clicking it in the panel, or by selecting it and clicking the Open button. Open *Example1.rule* within *Samples* in the same way.

Creating Files

New files can be created in Syntactica using the menu. Choose **File > New File** in the menu and select an appropriate file type from the **New File** submenu. An empty window of the appropriate type will open:



SHORTCUT! Open new windows using the Button bar:

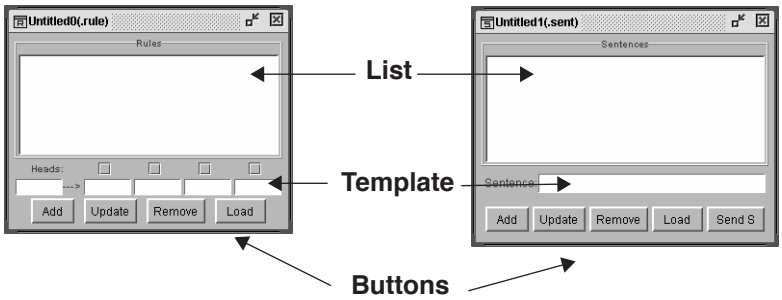


Entering Information

Once a new window is open, the user can enter information in it. Syntactica windows that communicate with TreeViewer have the same general layout. There is:

- a list portion,
- a template portion,
- a button portion.

The first lists the items in the window; the second and third display the contents of each item, and allow the user to edit those contents. For example, the Rule window and Sentence window look like this:



The user enters information in the template, clicks the Add button to create an item in the list. Lists of items are then edited and saved as files (see below).

EXAMPLE: Open a new Rules window and enter the following rule information in its template:

Click the Add button at the bottom of the window. The rule S → NP VP appears in the list, which now looks like this:

When the Add button is clicked, information disappears from the template. To display an item, select it in the list; its contents reappear in the template.

Each type of file requires its own kind of information to be entered. For more on the particular kind of information that is associated with each window, consult the individual chapters of this manual.

Editing Windows and Files

The contents of a window are edited using its buttons and standard text functions. Items in a list are deleted by selecting them in the list, and clicking the **Remove button**. Items in a list are edited by selecting them in the list, editing their contents in the templates, and clicking the **Update button**.

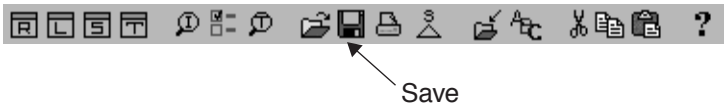
EXAMPLE: Click on the rule S → NP VP created in the previous example. Edit it in the rule template to look like this:

Click the Update button at the bottom of the window. The rule S → NP V now appears in the list.

Saving Files

To save the contents of a Syntactica window as a file, make that window the Key window and select **File > Save** from the menu (or enter `control s`). If the file is a new one, a Save documents Panel will appear.

SHORTCUT! Click on the Save icon in the Button bar:



Loading Files

Information that has been entered in a window must be loaded before Syntactica can use it. If you create some rules or lexicons, you must load them in order for Syntactica to work with them. If you change some rules or lexicons, you must reload these items in order for Syntactica to operate with the changes.

A Syntactica file is loaded (or reloaded) whenever:

- the file is opened, or
- the file is saved, or
- the Load button in its window is clicked.

Opening a file automatically causes it to be loaded into Syntactica. Saving from a window causes its contents to be loaded into Syntactica and a file containing those contents to be created or saved. Clicking the Load button in a window causes its contents to be loaded into Syntactica, without creating or saving a file.

Loading Sentence & Tree Windows

Sentence and Tree windows have a special status with respect to loading. Loading a sentence or tree window creates a link between it and TreeViewer. This link allows sentences or trees to be sent back and forth between the window and TreeViewer. For more information, see *Chapter 7 Sentence Files, Tree Files, and Grammars*.

Active Windows

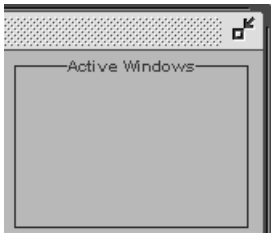
Windows whose contents have been loaded into Syntactica are called **active windows**.

EXAMPLE: Suppose you create a new set of rules in a Rules window and save them as a file. They are loaded into Syntactica and the window becomes the **active Rule window**.

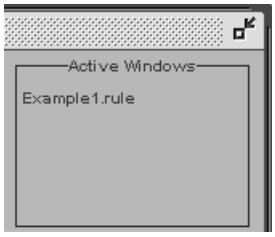
EXAMPLE: Suppose you create a new set of rules in a Rules window, and click its Load button. The rules are loaded into Syntactica and the window becomes the **active Rule window**. But the rules are not saved as a file.

EXAMPLE: Suppose you open an existing rule file. Its contents are loaded into Syntactica and it becomes the **active Rule window**.

Titles of all active windows are registered in the **Active Windows list**, located in the upper right-hand corner of TreeViewer:



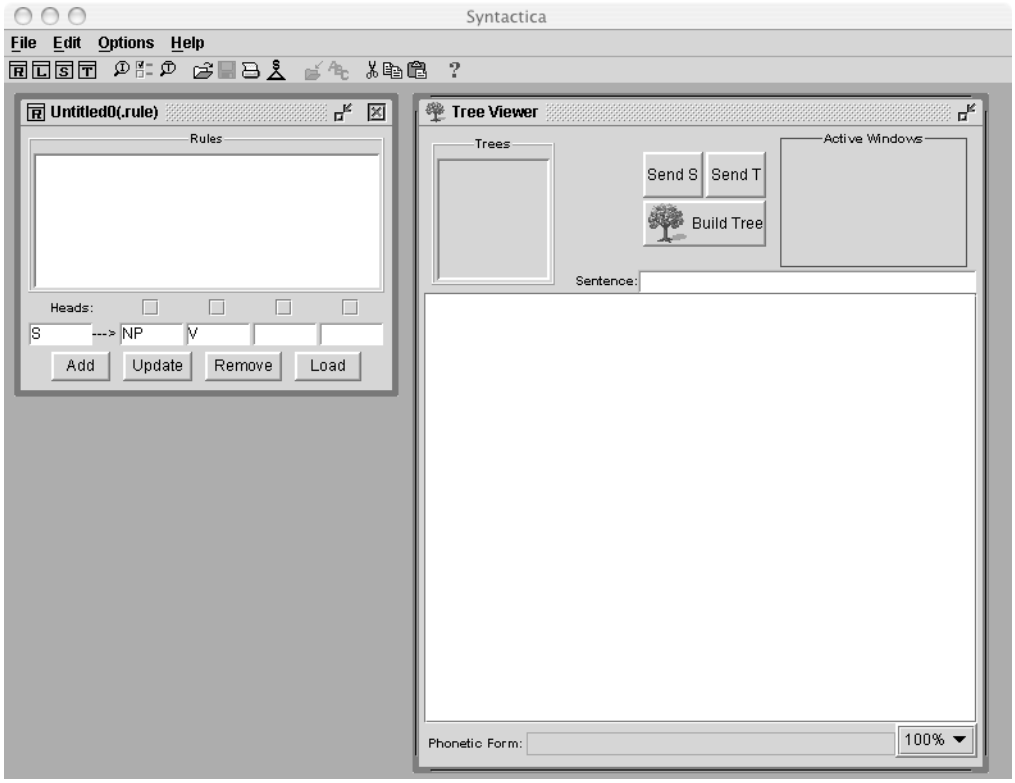
EXAMPLE: Suppose *Example1.rule* is opened. It is loaded and becomes the active Rules window. *Example1.rule* appears in the list:



Since multiple windows of a given type may be open at the same time, the active windows list is helpful in keeping track of what rules and lexicons are being used to build phrase markers, or what sentences and tree windows are linked to TreeViewer.

Sample Session: Building Phrase Markers

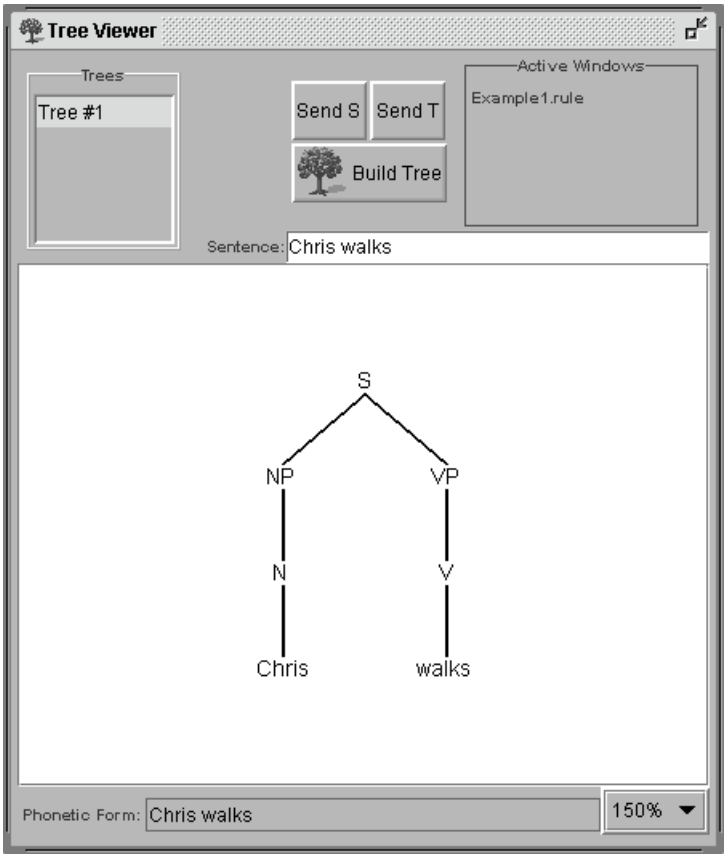
Launch Syntactica. Your screen should look something like this.



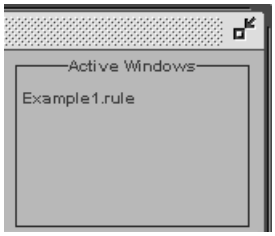
In the upper left corner is the application menu. Below it is a Button bar with shortcuts. On the screen there is:

- a TreeViewer window,
- a Rule window.

Choose **File > Open File > Rules** from the menu. Locate *Example1.rule* in the *Samples* folder and open it. Click in the sentence field of Tree-Viewer and enter the sentence *Chris walks*. Click the Build Tree button (or `press Return`). Syntactica generates the tree shown below.



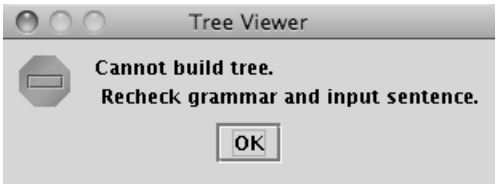
Note that *Example1.rule* is listed in the Active Windows list:



This is the tree assigned to the sentence *Chris walks* by the rules in *Example1.rule*.

You can enter any sentence you like as input. But Syntactica will succeed in building a tree only if the input sentence is within the scope of the rules (and lexicon) that have been loaded.

EXAMPLE: With *Example1.rule* loaded, edit your input sentence to *The man walks*. Click Build Tree (or `press Return`). Syntactica gives an error message:

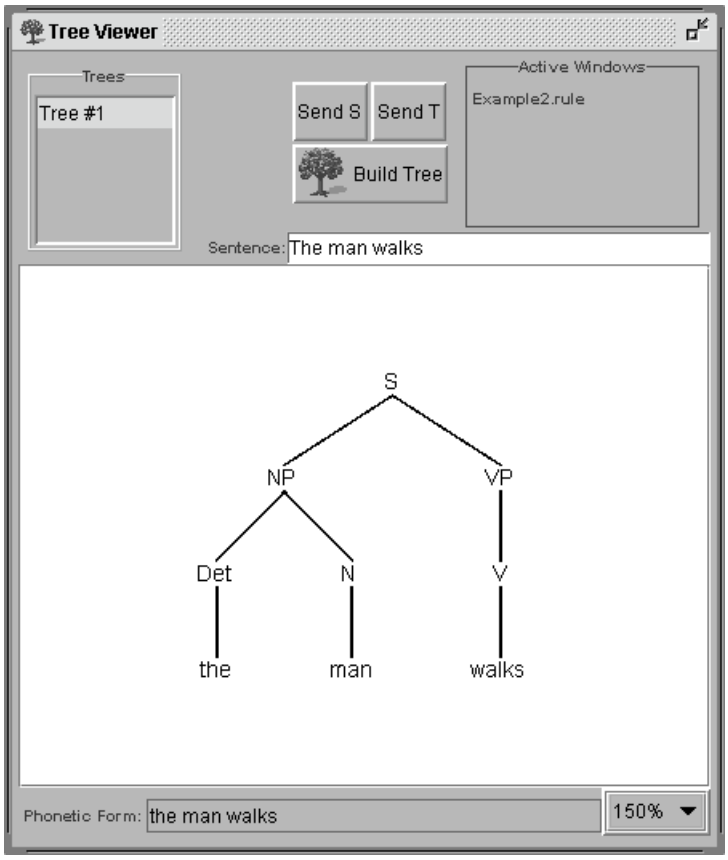


The sentence *The man walks* cannot be generated because rules for *the man* are not present in *Example1.rule.*

Changing Files

In the process of analyzing a given sentence, the user can open and load alternative files to see what phrase markers they yield.

EXAMPLE: Leaving *The man walks* as input sentence, open the rule file *Example2.rule*. Confirm that it is loaded into TreeViewer by inspecting the Active Windows list. Click the Build Tree button (or `press Return`). Syntactica now succeeds in generating a tree for *The man walks*:

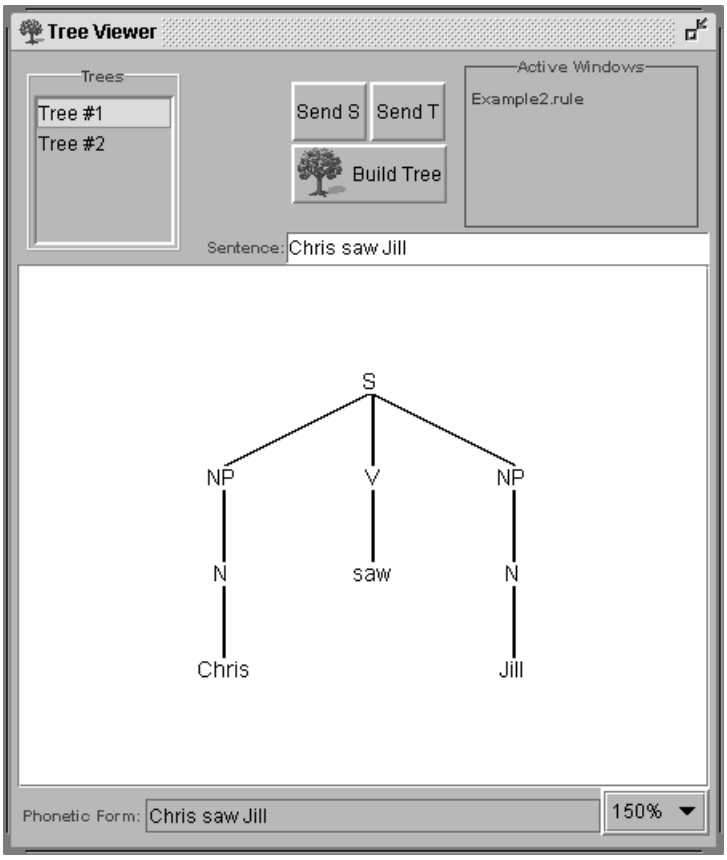


The build succeeds because *Example2.rule* (unlike *Example1.rule*) contains the resources to accommodate *the man*.

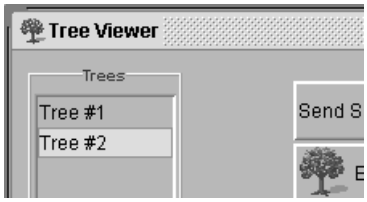
Viewing Alternative Structures

A given expression may have several different possible trees under a given set of rules (and lexicon). In this circumstance, Syntactica builds all possible trees and allows the user to view them sequentially.

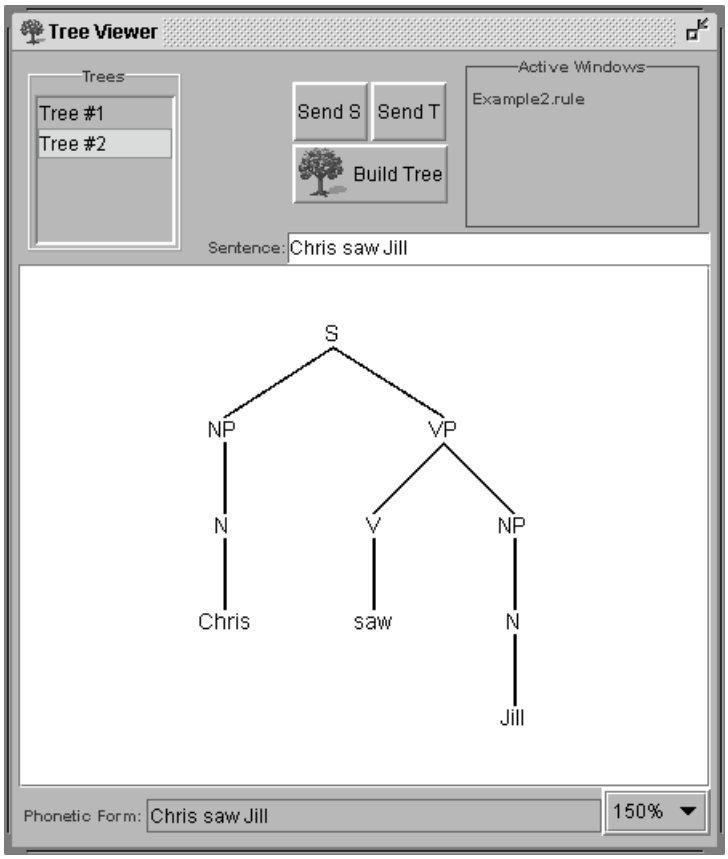
EXAMPLE: With *Example2.rule* loaded into TreeViewer, enter the input sentence *Chris saw Jill*, and build its tree.



TreeViewer displays one structure and indicates another in the **Tree list** in the upper left-hand corner. Tree #1 is highlighted in the list, showing that it is the one currently displayed. Click *Tree #2* in the Tree list.



The entry is highlighted and a new tree is displayed. This is Tree #2 for *Chris saw Jill* as generated by *Example2.rule*:



Clicking back and forth in the tree list allows you to rapidly view and evaluate structural differences in parses.

Using Rules and a Lexicon Together

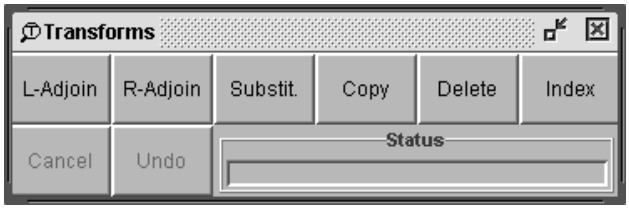
The example trees above were generated using rules alone. The procedure for building trees with rules and a lexicon is similar.

EXAMPLE: Open *Example3.rule* and *Example1.lex*. The Active Window list displays the titles of both files. Enter *Chris walks* and build its tree. Note that the structure is the same as the one assigned by *Example1.rule*. The

grammar consisting of *Example1.rule* alone is strongly equivalent to the grammar consisting of *Example3.rule* and *Example1.lex* together. It succeeds in building trees for the same input sentences, and assigns them the same tree structures.

Sample Session: Transforming Phrase markers

Structures generated by Syntactica can be transformed by various syntactic operations, including movement, copying, deletion, and indexing. These operations are performed using the Transforms panel. To get the Transforms panel, select **Options > Transforms** (or `enter control T` from the keyboard):



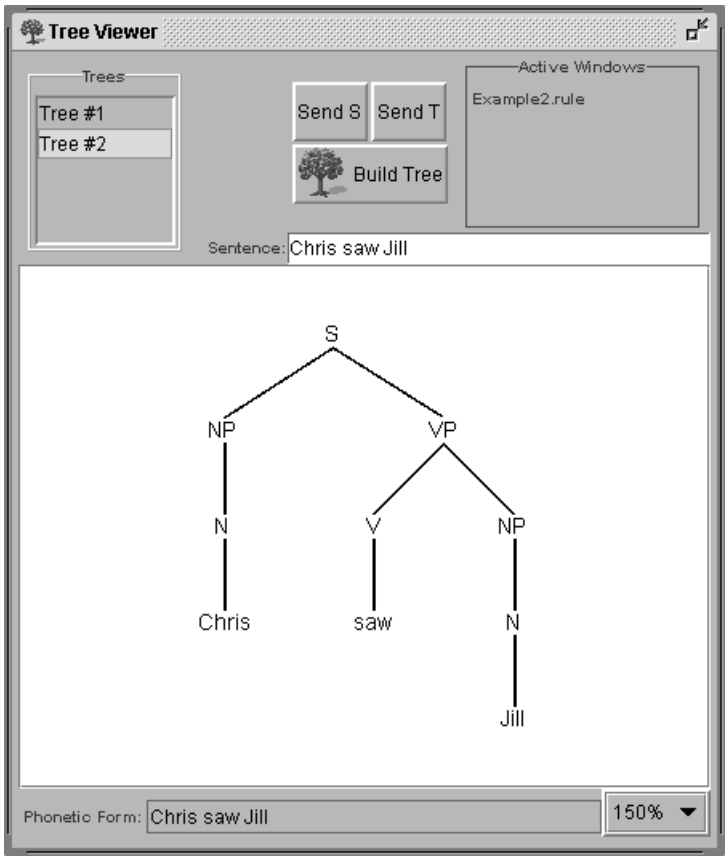
In the upper row are buttons for six syntactic operations. In the lower row is a Cancel button, which aborts an operation in progress, an Undo button, which undoes a completed operation, and a status field, which informs the user of where he/she is in an operation.

SHORTCUT! Click the Transforms icon in the Button bar:

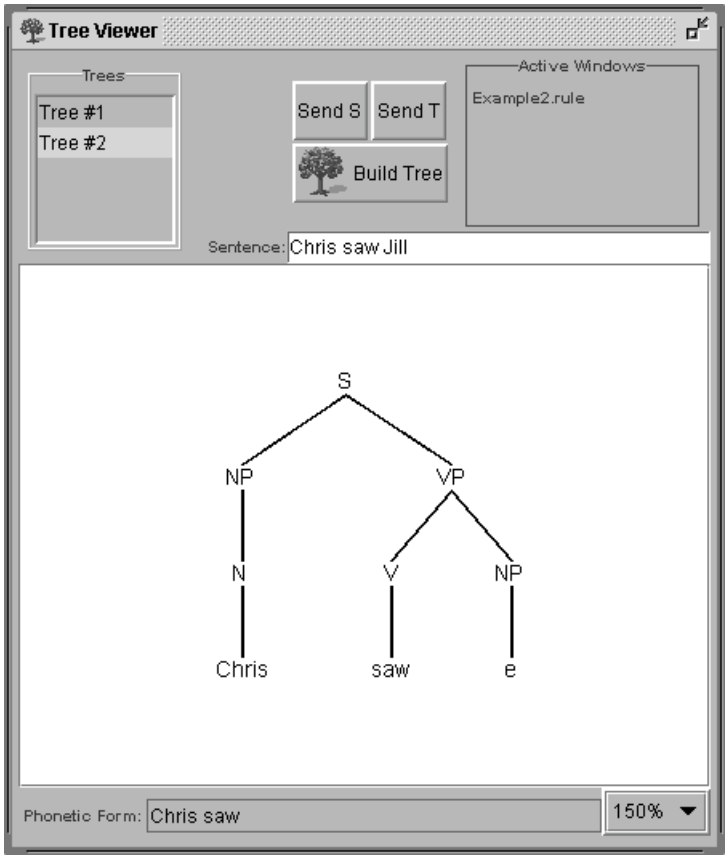
Performing Operations

To transform a tree, the user clicks on a node (or nodes) in it, and then clicks an operation button in the Transforms panel.

EXAMPLE: Open *Example2.rule*. Enter the sentence *Chris saw Jill* and click Build Tree. Select Tree #2. Open the Transforms panel. In the Tree-Viewer window, click on the NP node above *Jill*.

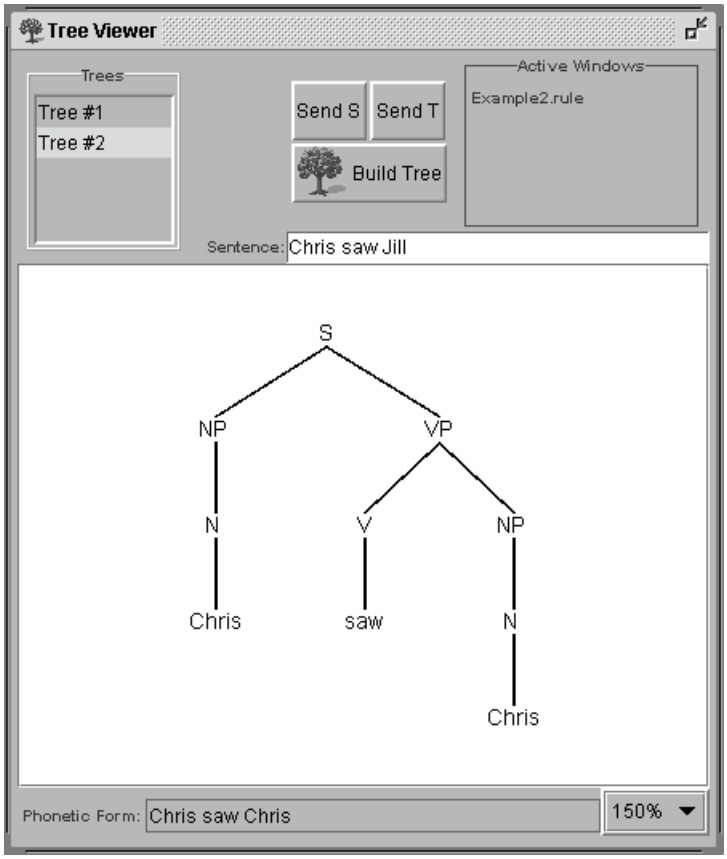


The node is highlighted showing that it has been selected. Now click the Delete button in the Transforms panel. The structure below NP is deleted and replaced by the empty string *e*:



In a deletion, a **target node** is selected ([$_{NP}$[$_{N}$ *Jill*]]) and an operation is performed on it.

EXAMPLE: Take the tree just created by deletion and click on the NP node [$_{NP}$[$_{N}$ *Chris*]]. The node is highlighted. Click the **Copy** button in the Transforms panel. Now click on the node [$_{NP}$ *e*]. The contents of [$_{NP}$[$_{N}$ *Chris*]] are copied into [$_{NP}$ *e*]. The result is a tree with the terminal string *Chris saw Chris*.



In a copying, a **source node** ([$_{NP}$[$_{N}$ *Chris*]]) is selected in addition to a **target node** ([$_{NP}$ *e*]). The operation is performed on the pair (the first node is copied to the second).

Undoing a Transformation

The **Undo** button in the Transforms panel allows you to undo the last transformation performed. Thus, if Undo is clicked after the delete operation, the deleted item is restored. If Undo is clicked after the copying operation, the pre-copy tree is restored.

USER NOTE: If you have performed a number of operations on a tree, and wish to restore the original tree, simply click the Build Tree button again. The input sentence is reparsed and its trees are redisplayed.

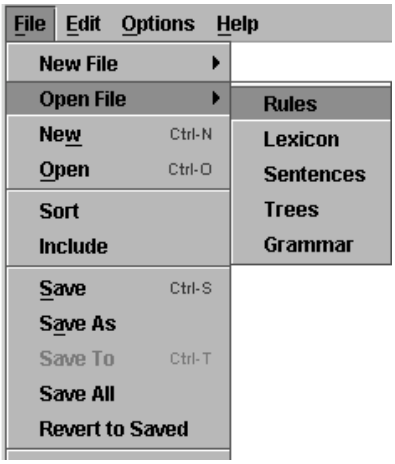
CHAPTER 4

Working with Rules and Rule Files

A rule file consists of a set of context-free phrase structure rules, together with a specification of the head(s) of those rules. Rule files have the extension *.rule*.

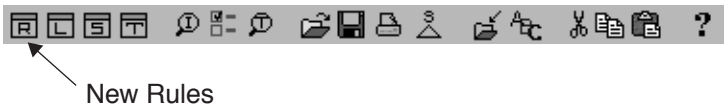
The Rule Window

Rule files are created, displayed and edited in rule windows. A rule window appears whenever you open an existing rule file, or create a new one. To open an existing rule file choose **File > Open File > Rules** from the main menu.



To create a new rule file choose **File > New File > Rules**.

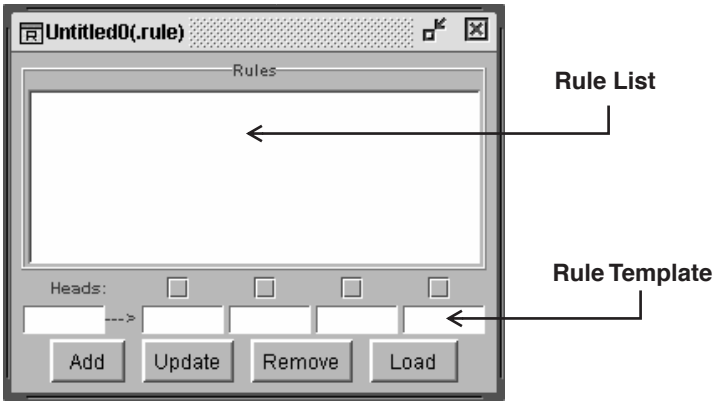
SHORTCUT! Open new rule windows with the Button bar:



The Parts of a Rule Window

Rule windows consist of two major parts:

- a rule list, for collecting and displaying rules,
- a rule template, for creating and editing rules.



The Rule List

The rule list is a scrollable area where rules are displayed.

The Rule Template

The rule template contains five text fields. The left most is separated from the others by an arrow (→). Above each of the four right-hand fields is a head box, for specifying whether the item in the field is a head. Below the rule template are buttons for adding rules to the list, updating, or removing them. There is also a Load button for loading rules into TreeViewer.

Whenever you select a rule in the rule list by clicking on it, that rule is highlighted in the list and displayed in the rule template.

Entering Rules

Rules are entered using the rule template, according to the following conventions:

- The category of the mother node is entered in the single field to the left of the arrow.
- The categories of the daughter nodes are entered in the four fields to the right of the arrow.
- Daughters are entered from left to right, beginning with the first field and leaving no empty fields in between.

The third convention applies specifically to head-initial languages like English. Entering rules for head-final languages like German or Japanese follows different conventions. For more information, see *Chapter 11 Head-Final Languages*.

Things to Keep in Mind

There are a couple of points to keep in mind when entering rules:

- Syntactica is case-sensitive, which means that S → NP VP and S → Np VP are treated as separate rules.
- Any string of characters that can be typed from the keyboard can be entered as a category label. But no spaces should be left between the characters in a string. In fact the program will not allow spaces to be entered.

EXAMPLE: The rule S → NP VP would be entered like this:

In this rule, you can type 'VP' in the middle field, but not 'V P'.

Adding, Editing, Saving and Printing Rules

Rules that have been entered in the rule template can be added to the rule list, edited (if necessary), saved as a file and printed.

Adding Rules to the Rule List

To add a rule to the rule list:

1. Enter the rule in the rule template.
2. Click the Add button (or type Return).

Editing Rules in the Rule List

To edit a rule in the rule list:

1. Select the rule in the list.
2. Make changes in the template with the usual text functions.
3. Click the Update button (or type Return).

Deleting Rules from the Rule List

To delete a rule from the rule list:

1. Select the rule in the list.
2. Click the Remove button.

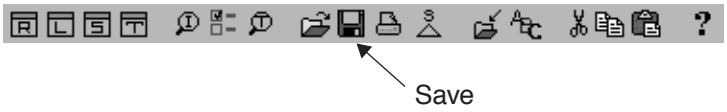
Saving Rules in the Rule List

To save a list of rules as a file:

1. Make the rule window the key window by clicking on it.
2. Choose **File > Save** from the menu (or `enter control s`).

If the file is a new one, a Save Rule Documents panel appears asking you to name the file and give it a location.

SHORTCUT! Click the Save button in the Button bar:

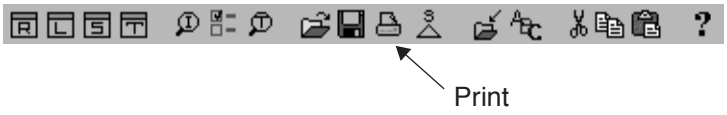


Printing the Rule List

To print a list of rules:

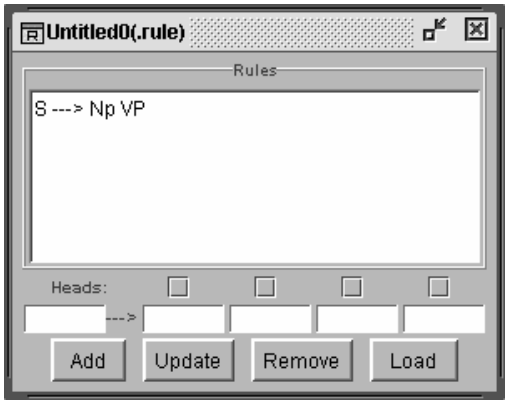
1. Make the rule window the key window.
2. Select Print from the menu (or `enter control p`).

SHORTCUT! Click the Print button in the Button bar:



When a list of rules is printed, the head of each rule (if any) appears in boldface.

EXAMPLE: Select **File > New File > Rules**. A new rule window opens (entitled *Untitled*). Click in the left most field in the rule template and enter the symbol 'S'. Now `press Tab`. The cursor moves rightward to the first field after the arrow. Enter 'Np', and `press Tab` again. Enter 'VP' and click the Add button. The rule appears as the first line in the rule list. The window *UNTITLED* should look like this:



Rules (in a head-initial language) are entered from left to right beginning with the first field and leaving no empty fields in between. Practice by adding the following rules to the list:

NP → N

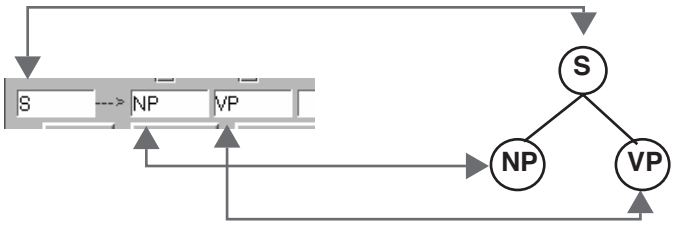
NP → *the* N

VP → V

VP → N

Click on the rule VP → N in the rule list. It is highlighted in the list and appears in the rule template. Click Remove. VP → N disappears from the list. Click on the rule S → Np VP. Change 'Np' to 'NP' in the rule template. Click Update. S → Np VP changes to S → NP VP in the rule list. With the *UNTITLED* window as key window, select **File > Save** (or `enter control s`). Save the list of rules as the file *MyRules*.

Rules correspond closely with trees. The category before the arrow will correspond to the mother node. The first category after the arrow will correspond to the left most daughter in the tree diagram; the next category in the rule will correspond to the next daughter to the right; and so on:

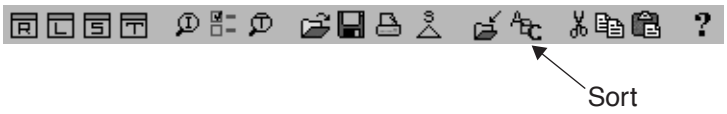


Sorting Rules

Lists of rules can be sorted so that rules of the same category are grouped together. To do so:

1. Make the window to be sorted the key window.
2. Choose **File > Sort** from the application menu.

SHORTCUT! Click the Sort icon in the Button bar:

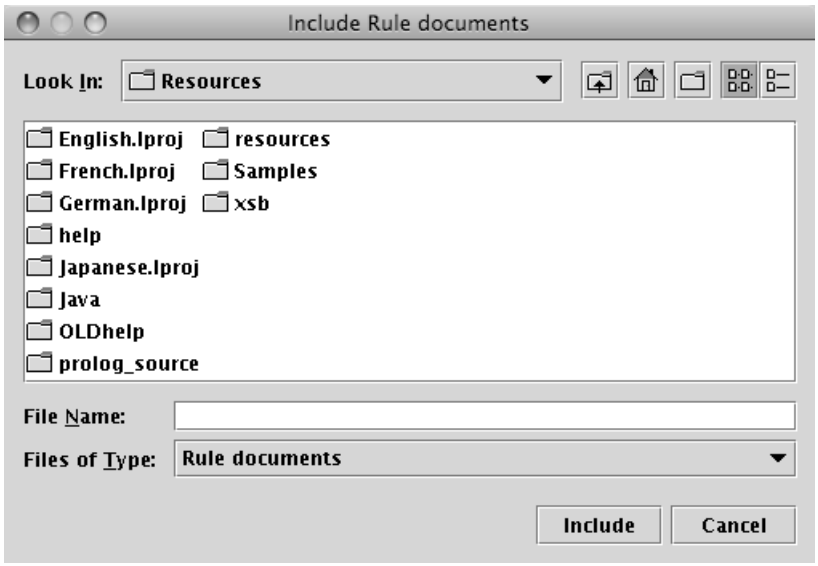


Sorting alters the contents of a window, just like adding or deleting rules, so remember to save after sorting!

Including Rules From Other Rule Files

Rules from other rules files can be added to those in any open rule window using the Include command. To do so:

1. Make the window to be added-to the key window by clicking on it.
2. Choose **File > Include** from the menu (or `enter control i`). A panel appears asking you to select the rule file whose content is to be included.



3. Select the file of rules to be included.
4. Click Include (or `press Return`).

SHORTCUT! Click the Include icon in the Button bar:

Include not only imports rules into the current file, but also adds certain information specified in the Rules Inspector panel. Specifically:

- Bounding nodes from the included file are added to those of the current window
- Variable categories from the included file are added to those of the current window

Include does not add root node or default head direction choice as this would potentially involve changing the values in the current window. For more on the Rules Inspector panel, see the section *Setting Rule Defaults* later in this chapter.

USER NOTE: Include is useful for combining results in separate files, and in allowing you to take results from previous work and extend them. As an alternative to copying a rule file and changing its title, you may simply include its contents into an empty rule window.

Setting and Changing Heads

The user may specify which node(s) in a phrase structure rule is (are) the head(s) of that rule. To do so:

1. Enter the rule (or select it in the rule list).
2. Click in the head boxes above the appropriate nodes in the rule template.

EXAMPLE: The following indicates that the node N is a head in the rule NP → the N:

Head information is not shown explicitly in the rule list. To inspect the head(s) for a given rule, select it so that it is displayed in the rule template.

To change the head(s) of a rule:

1. Select the rule in the rule list.
2. Click in the box(es) corresponding to the old head node(s) to remove the old check(s).
3. Click in the box(es) corresponding to the new head node(s).
4. Click the Update button.

Marking a node as a head instructs Syntactica to pass its syntactic features to its mother node (the node on the left-hand side of the arrow). If no head is selected for a rule, no feature-passing occurs. If two heads are selected, feature passing from both nodes will occur. Etc. For more on this, see *Chapter 10 Heads, Features, and Complements*.

Setting Rule Defaults

The user may specify defaults in the creation of rules and rule files. These include:

- root node,
- default head direction,
- variable categories,
- bounding nodes for Subjacency.

To set defaults for a rule window, click in its window and select **Options > Inspector** from the menu. The Rules Inspector panel appears:

SHORTCUT! Open the Inspector panel with the Button bar:

The Root Node

The **Root Node** is the symbol that Syntactica places at the top of the tree that it builds. The initial default root node symbol in Syntactica is S, the

usual category of sentences. However the root symbol can be reset to any category.

To reset the root node:

1. Edit the root node field using the usual text editing functions.
2. Click Set (or `press Return`).
3. Save the rules or load them into Tree Viewer.

EXAMPLE: Open and load *Example1.rules*. Open the Inspector and reset the root node as VP. Save the file. Now enter *walks* in the sentence field of TreeViewer and click Build Tree. Syntactica generates a tree consisting of VP node dominating a V node dominating the lexical item *walks*.

USER NOTE: Being able to generate trees with root nodes other than S is handy for debugging. When a build fails, one way to locate the problem is to attempt to build sub constituents of the input expression. To do this, the user must change the root symbol to the category of the sub constituents (NP, VP, PP, etc.).

Head Direction

The Rule Inspector allows you to set a default head choice for a file so that whenever rules are created in that file, they are automatically specified as being Head Initial, Head Final, or as having No Head. No Head is the initial default for Syntactica.

To set or alter default head direction:

1. Click the pop-up button in the Rules Inspector panel.
2. Drag to the desired choice and release.
3. Click Set (or `press Return`).
4. Save the rules or load them into TreeViewer.

In addition to root node and head direction settings, the Inspector panel allows the user to declare variable categories and bounding nodes for Subjacency. For more information on variable categories and Subjacency, see *Chapter 9 Constraints on Transformations*.

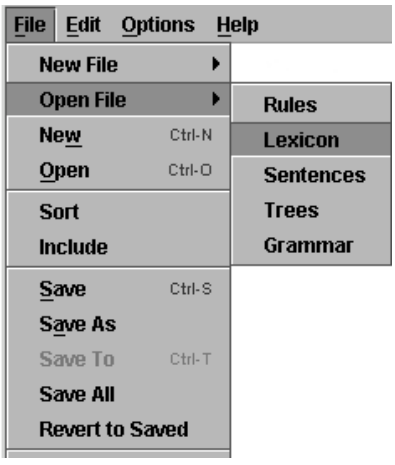
CHAPTER 5

Working with Lexicons and Lexicon Files

A lexicon file consists of a set of lexical items, or words, together with a specification of their properties, including syntactic category, features, complements and complement features, and audibility. Lexicon files have the extension *.lex*.

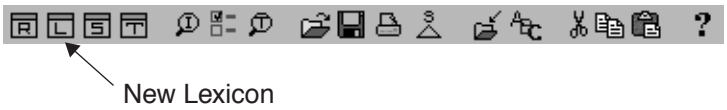
The Lexicon Window

Lexicon files are created, displayed and edited in lexicon windows. A lexicon window appears whenever you open an existing lexicon file, or create a new one. To open an existing lexicon file, choose **File > Open File > Lexicon** from the main menu.



To create a new lexicon file choose **File > New File > Lexicon**.

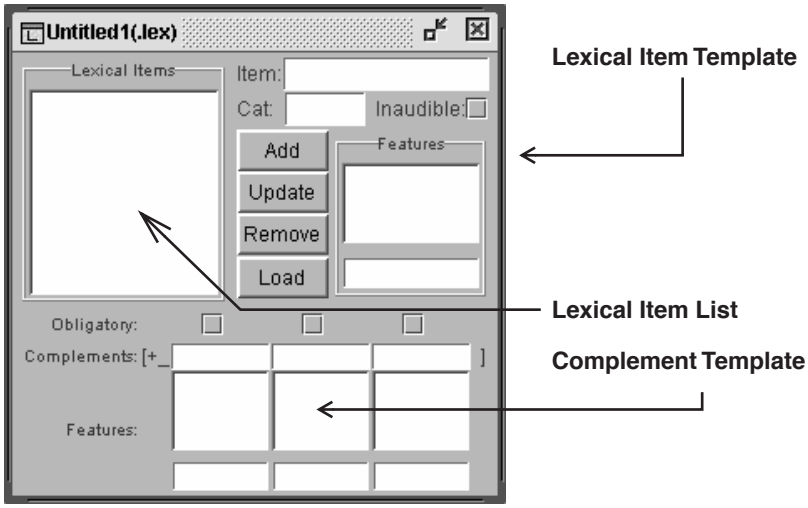
SHORTCUT! Open new lexicon windows with the Button bar:



The Parts of a Lexicon Window

A Lexicon window consists of three major parts:

- a lexical item list, which displays the words (or morphemes) in a file,
- a lexical item template, for entering words and their features,
- a complement template, for specifying complements and their features.

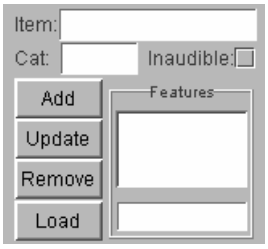


The Lexical Item List

The lexical item list is a scrollable matrix in which lexical items are displayed.

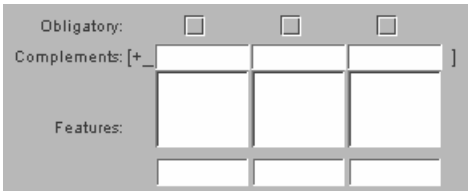
The Lexical Item Template

The lexical item template contains an item field, a category field, and a Features area. It also contains a box marked “Inaudible” for specifying whether a word is silent (unpronounced) or overt (pronounced). There are buttons that allow you to add words to the list, update them, or remove them. There is also a Load button for loading the lexicon into TreeViewer.



The Complement Template

The complement template contains three main fields, each with its own Features area, and each with its own box for indicating whether the complement is obligatory or optional.



When you select a word in the lexical item list, it is highlighted in the list and displayed in the lexical item template. In addition, its complements and their features are displayed in the complement template.

Entering Lexical Information

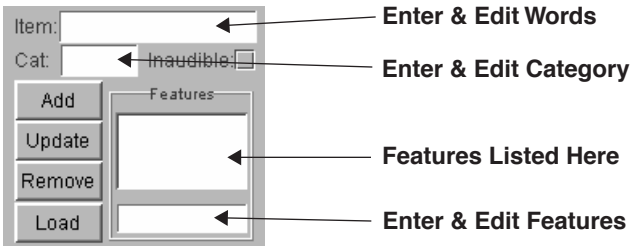
Information about lexical items is entered in the lexical item template and the complement template of a lexicon window.

The Lexical Item Template

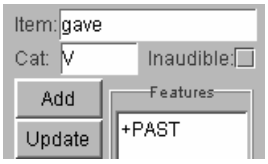
The lexical item template is used to enter the following information about a word:

- its spelling,
- its syntactic category,
- whether it is inaudible,
- its syntactic features (if any).

Items and their category are entered and edited in the appropriate fields. Features are entered and edited in the Features area. To enter a feature, type it in the lower field and type Return. To edit a feature, select it in the upper field, edit it in the lower field, and type Return:



EXAMPLE: The past tense form of the English verb *give* might be entered as follows:



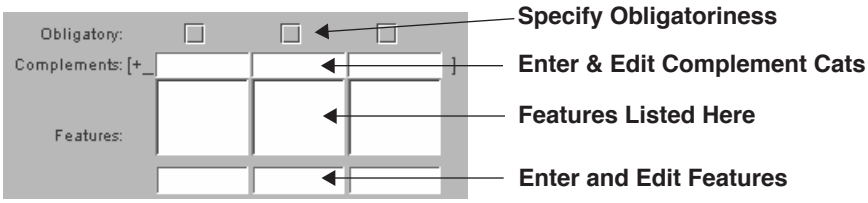
USER NOTE: Spelling and syntactic category must be entered for all lexical items. Otherwise, Syntactica returns an error message:

The Complement Template

The complement template is used to enter the following information about a word:

- the category of its complements,
- the obligatoriness or optionality of its complements,
- the syntactic features of its complements (if any).

The basic picture is this:



Category

Complements are specified by category in the three complement fields. They are entered from left to right, beginning with the first field, and leaving no empty fields between entries.

Obligatoriness

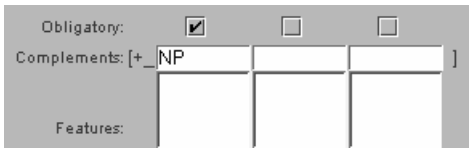
Obligatoriness of a complement is indicated by clicking the box above it. Check marks are understood by Syntactica as followings:

- A checked box above a nonempty field means that a complement of the relevant category (and features) must appear
- An unchecked box above a nonempty field means that a complement of the relevant category (and features) mayappear, but complements of any other categories and featuresmust not appear
- An unchecked box above an empty field means that no complement of any category and features may appear

Features

Complement features are entered and edited as with lexical item features.

EXAMPLE: The complement template below for *devoured* specifies that this verb must be followed by an NP object:

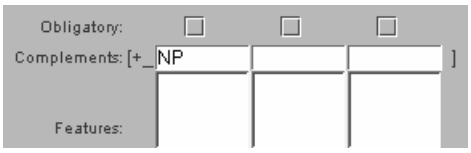


This yields the distribution (where the asterix '*' indicates an ill-formed sentence):

*Jill devoured

Jill devoured [$_{NP}$ a pizza]

EXAMPLE: The complement template below for *ate* specifies that this verb may be followed optionally by an NP object, but no other category of complement may appear:

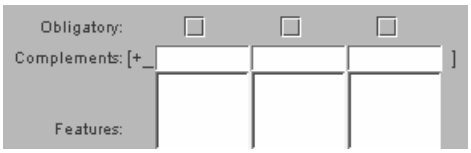


This yields the distribution:

Jill ate

Jill ate [$_{NP}$ a pizza]

EXAMPLE: The complement template below for *dined* specifies that the verb takes no complements at all:



This yields the distribution:

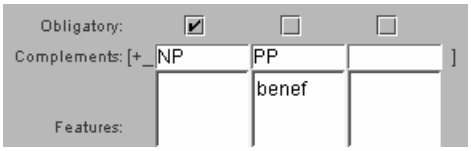
Jill dined

*Jill dined [$_{NP}$ a pizza]

Multiple Complements

These conventions interact in the expected way when more than one complement is declared.

EXAMPLE: The complement template below for *baked*, specifies that the verb must have an NP object, and may take a following PP with the feature *benef*. No other kind of complement may follow the NP, however.



Assuming PPs headed by *for* bear the feature *benef,* this yields the distribution:

*Chris baked

Chris baked [$_{NP}$ a cake]

Chris baked [$_{NP}$ a cake][$_{PP}$ for Jill]

*Chris baked [$_{PP}$ for Jill] [$_{NP}$ a cake]

*Chris baked [$_{NP}$ a cake] [$_{AP}$ tall]

*Chris baked [$_{NP}$ a cake] [$_{PP}$ to Jill]

USER NOTE: These conventions for entering complement information apply to head-initial languages like English. Lexicons for head-final language like German or Japanese follow different conventions. For more information, see *Chapter 11 Head-Final Languages*.

Adding, Editing, Saving and Printing Lexicons

Lexical items that have been entered in the templates can be added to the item list, edited (if necessary), saved as a file and printed.

Adding Lexical Items

To add a lexical item to the lexical item list:

1. Enter the item in the templates.
2. Click the Add button.

Editing Lexical Items

To edit a lexical item in the lexical item list:

1. Select the item in the list.
2. Make changes in the templates with the usual text functions.
3. Click the Update button.

Deleting Lexical Items

To delete a lexical item from the lexical item list:

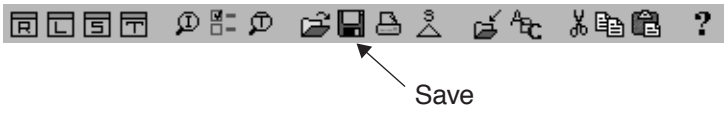
1. Select the item in the list.
2. Click the Remove button.

Saving Lexical Items

To save a list of lexical items as a file:

1. Make the lexicon window the key window.
2. Choose **File > Save** from the menu (or `enter control s`).

SHORTCUT! Click the Save button in the Button bar:

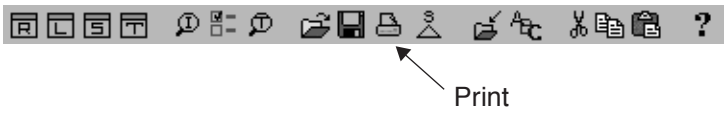


Printing Lexical Items

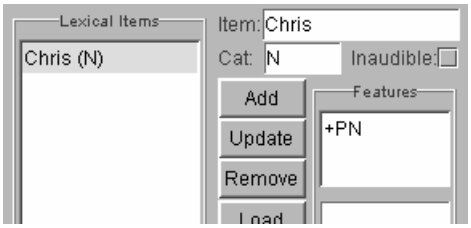
To print a lexicon:

1. Make the lexicon window the key window.
2. Select Print from the menu (or `enter control p`).

SHORTCUT! Click the Print button in the Button bar:



EXAMPLE: Select **File > New File > Lexicon**. A new lexicon window opens (entitled *Untitled*). Click in the lexical item template and enter the word *Chris*. Hit Tab. The cursor moves to the category field. Enter 'N' to mark the element as a noun. Tab again. The cursor moves to the lower feature field. Enter '+PN' to indicate that *Chris* is a proper noun and type Return. Click the Add button. The name in the template now appears as the first item in the lexical item list. Now select the word *Chris* in the lexical item list. The word is highlighted in the list and its properties are displayed in the template:



Practice by adding the following words to the file, including their category, and feature information:

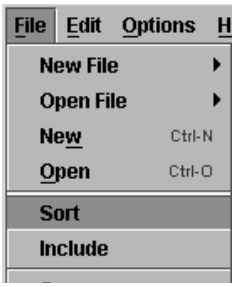
dog	N,	-PN
walks	V,	+PRES
ran	V,	-PRES
the	Det	

With the untitled lexicon window as the key window, select **File > Save** (or `enter control s`). Save the lexicon as *MyLex*.

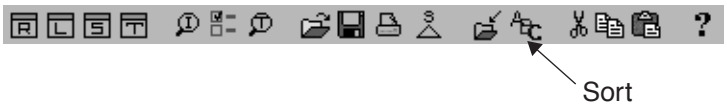
Sorting a Lexicon

Lists of lexical items can be sorted so that items of the same category are grouped together. To do so:

1. Make the lexicon window to be sorted the key window.
2. Choose **File > Sort** from the menu.



SHORTCUT! Click the Sort icon in the Button bar:

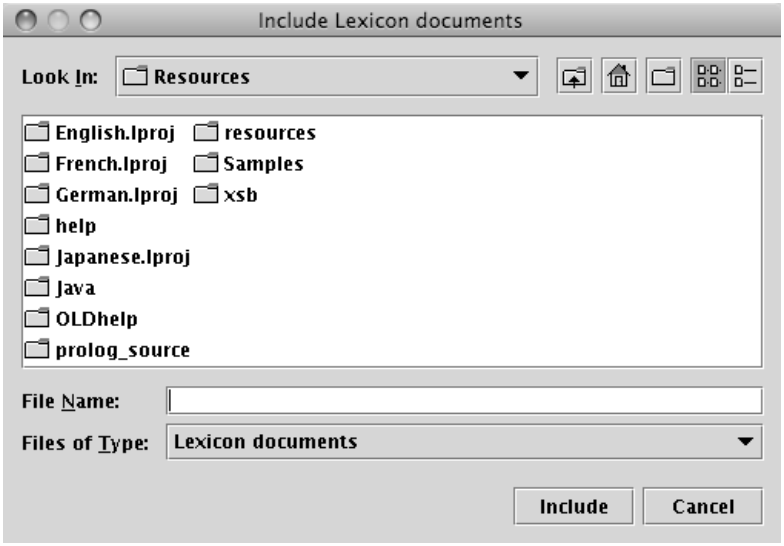


Sorting alters the contents of a window, just like adding or deleting items, so remember to save after sorting!

Including Lexical Items From Other Lexicon Files

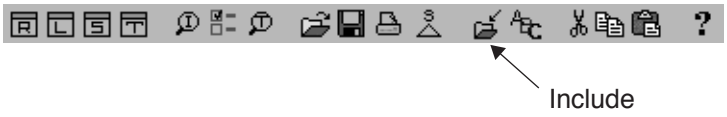
Lexical items from other lexicon files can be added to those in any open lexicon window using the Include command. To do so:

1. Make the window to be added-to the key window by clicking on it.
2. Choose **File > Include** from the menu (or `enter control i`). A panel will appear asking you to select the lexicon file whose contents are to be included.



3. Select the lexicon file to be included.
4. Click Include (or `press Return`).

SHORTCUT! Click the Include icon in the Button bar:



Include imports all lexical items from the selected file into the current file. Include also adds certain information specified in the Lexicon Inspector panel. Specifically:

> Pleonastic words from the included file are added to those of the current window.

The Include command does not add information about head direction as this would potentially involve changing the values in the current window. For more on pleonastic words, see *Chapter 9 Constraints on Transformations*.

USER NOTE: The Include command is useful for combining results in two different lexicon files, and in allowing you to take results from previous work and extend them. As an alternative to copying a given lexicon file and changing its title, simply include the file into an empty lexicon window.

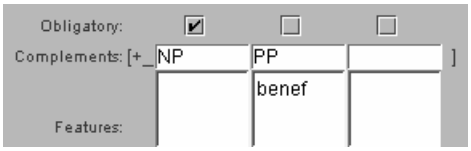
More on Features

Enforcing Co-occurrence Restrictions

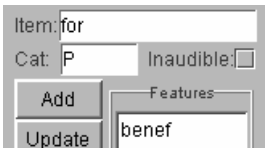
Features can be used in Syntactica to enforce co-occurrence restrictions between words and phrases.

EXAMPLE: Suppose we wish to specify the English verb *baked* as requiring any complement PP occurring after its direct object to contain the benefactive preposition *for*. This restriction can be captured in Syntactica using features. We enter:

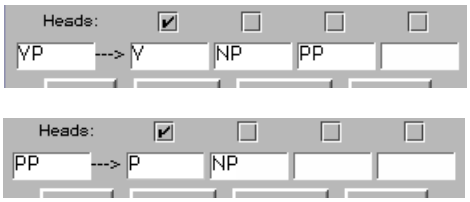
- the complement template for *baked* as shown:



- the lexical information of the preposition *for* (and only the preposition *for*) as shown:



- rules for VP and PP with headedness as shown:



Under these conditions, Syntactica will generate [$_{VP}$ V NP PP] structures with *baked* only when PP contains the preposition *for*.

Feature Values

Features are often analyzed as binary, taking one of the two values + or -. Furthermore, no single expression is allowed to have both + and - values for a given feature.

Syntactica implements both of these notions. When the user employs +F and -F as character strings for features (e.g., +dative and -dative, or +locative and -locative), Syntactica recognizes a relation between the two. A specification of +F and -F for a single lexical item is thus treated differently than a specification of +F and -G. Syntactica treats any feature not explicitly specified for a + or - value as having an implicit + value.

Syntactica will reject any derivation resulting in opposite values for the same feature being assigned to a single category. If the user assigns +F and -F (or F and -F) to a single lexical item α, then any derivation involving α will be excluded. Likewise, if the user assigns +F (or F) to a lexical item α, and -F to a lexical item β, and both features are inherited by a category γ, then this derivation will be excluded.

For more on the use of features, see *Chapter 10 Heads, Features and Complements*.

How Syntactica Distinguishes Lexical Items

Syntactica distinguishes lexical items as follows:

- Differently spelled items always receive distinct entries in a list
- Items of different category always receive distinct entries in a list

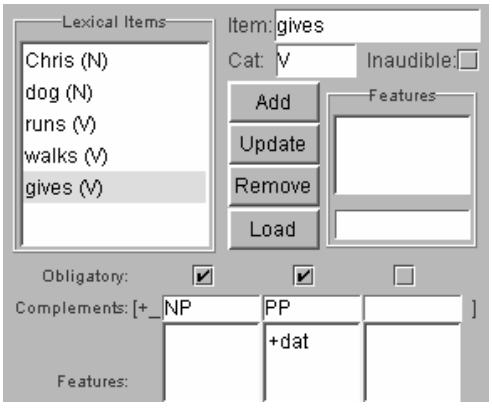
EXAMPLE: *Catalogue* and *catalog* would be assigned distinct entries even though they have identical pronunciation and meaning.

EXAMPLE: The noun (N) *look* and the verb (V) *look* would be assigned distinct entries even though they have identical spelling and closely related meanings.

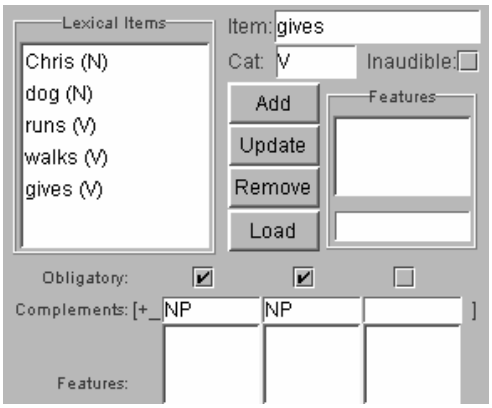
Multiple Entries

When the user tries to add a lexical item to a list that already contains a word of the same spelling and category, Syntactica prompts the user to decide whether this is a new item.

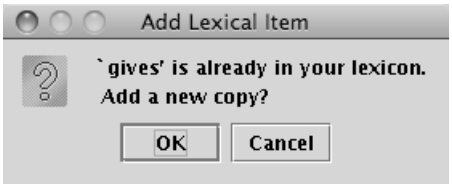
EXAMPLE: The verb *give* can appear with an object and a prepositional phrase complement, as in *Chris gives Fido to Jill*. Alternatively, it can appear with two objects, as in *Chris gives Jill Fido*. Suppose an entry for *gives* with its [+__ NP PP] complement frame is present in a lexicon:



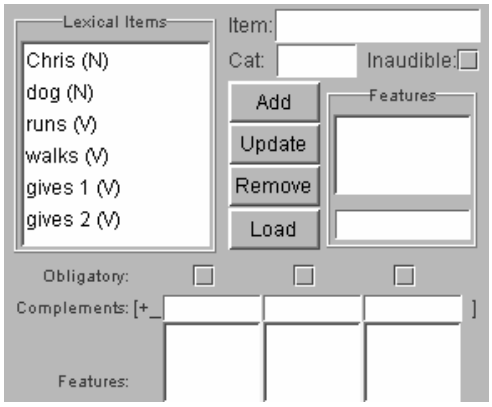
The user now attempts to add an entry for *gives* with its [+__ NP NP] complement frame:



The following panel appears:



If the user elects to add, the new item appears in the lexicon list with a numerical index, showing that it is one of several items of that spelling and category.



If the user does not elect to add, the operation is canceled.

Lexicons and Dictionary Entries

The behavior of Syntactica in distinguishing lexical items accords fairly well with standard dictionary practice. Dictionaries typically assign items of identical spelling but distinct category to separate lexical entries. On the other hand, differences of complement selection are usually treated as syntactic variation within a single lexical entry.

EXAMPLE: *Webster's Ninth New Collegiate Dictionary* assigns *brand* the noun (n) and *brand* the verb (vt) to two separate entries:

[1]brand \'brand\ *n*
[ME, torch, sword, fr. OE; akin to OE *bærnan* to burn]
(bef. 12c)
1a: a charred piece of wood
b: FIREBRAND 1
c: something (as lightning) that resembles a firebrand
2: SWORD
3a (1)**:** a mark made by burning with a hot iron to attest manufacture or quality or to designate ownership
(2)**:** a mark made with a stamp or stencil for similar purposes**:** TRADEMARK
b (1)**:** a mark put on criminals with a hot iron
(2)**:** a mark of disgrace**:** STIGMA ⟨the *brand* of poverty⟩
4a: a class of goods identified by name as the product of a single firm or manufacturer**:** MAKE
b: a characteristic or distinctive kind**:** VARIETY ⟨a lively *brand* of theater⟩
5: a tool used to produce a brand

[2]brand *vt*
(15c)
1: to mark with a brand
2: to mark with disapproval**:** STIGMATIZE
3: to impress indelibly ⟨*brand* the lesson on his mind⟩
— brand•er *n*

By contrast, the examples included for '*give* vt.' (*give a doll to a child* and *the law gives citizens the right to vote*) show that the latter is considered a single entry, despite having two different complement frames:

[1]give \'givvb\ **gave** \'gāv\; **giv•en** \'giv-ən\; **giv•ing**
[ME *given*, of Scand origin; akin to OSw *giva* to give; akin to OE *giefan, gifan* to give, L *habē* to have, hold]
vt
(13c)
1: to make a present of ⟨*give* a doll to a child⟩
2a: to grant or bestow by formal action ⟨the law *gives* citizens the right to vote⟩

Indexed items in a Syntactica lexicon (such as the two *give*'s in the preceding example) can be viewed as corresponding to subentries of a single entry, whereas separate, unindexed items correspond to distinct lexical entries.duck

CHAPTER 6

TreeViewer

TreeViewer displays syntactic trees generated by Syntactica. Generating trees requires the user to:

1. Load rules and (typically) a lexicon
2. Input a sentence or other expression
3. Instruct Syntactica to build a tree for the sentence

The TreeViewer window is also used to transform trees by means of the Transforms panel.

The TreeViewer Window

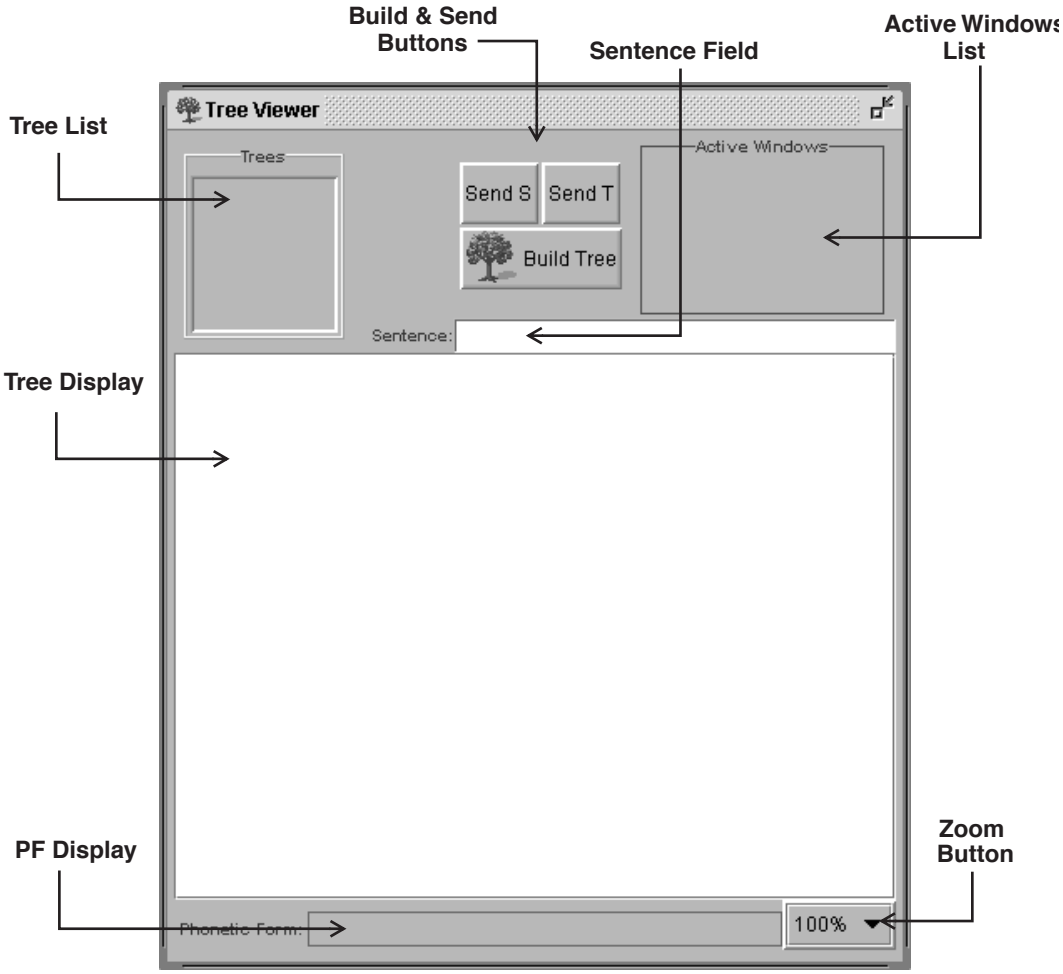
A TreeViewer window appears when Syntactica is launched and stays open while Syntactica is in use. It can be miniaturized, but it cannot be closed.

The Parts of the TreeViewer Window

The TreeViewer window consists of several elements:

- a Trees List,
- a Tree Display,
- a PF (Phonetic Form) Display,
- Zoom buttons,
- a Sentence field,

- an active windows list,
- Build & Send buttons.



The Tree List

The Tree List shows how many trees have been generated for an input expression, and which trees is currently displayed.

The Tree Display

The Tree Display exhibits trees that have been built for the input expression using the currently active rule and lexicon files.

The PF(Phonetic Form) Display

The PF Display gives the surface expression corresponding to the trees in the Tree Display.

The Sentence Field

The Sentence Field is where the user enters an input expression whose tree is to be built.

The Active Windows List

The Active Windows list shows which rules and lexicons are currently in use by Syntactica to build trees. It also shows which sentence and tree windows TreeViewer can send information to using its Send S and Send T buttons. For more information on Sentence and Tree windows, see *Chapter 7 Sentence Files, Tree Files, and Grammars*.

The Zoom Button

The Zoom button is used to resize the tree in the tree display.

The Build & Send Buttons

The Build and Send buttons are used to instruct Syntactica to generate a phrase marker and to send information to active sentence and tree windows, respectively.

Loading Rules and Lexicons

Loading rules and lexicons into TreeViewer makes them available for building trees. There are three different ways in which a set of rules or a lexicon may be loaded into TreeViewer:

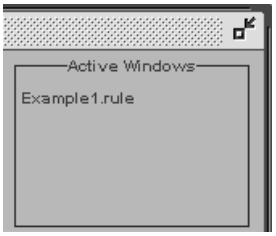
- by opening an existing file,
- by saving,
- by clicking the Load button.

Opening a rule or lexicon file automatically causes that file to be loaded into TreeViewer. Saving a set of rules or a lexicon causes a file containing those rules or items to be created and loaded into TreeViewer. Clicking the Load button in a rule or lexicon window causes its rules or lexical items to loaded into TreeViewer, without creating or modifying a file. This allows the user to experiment with rules and lexicons without accumulating files.

The Active Windows List

Whenever the contents of a rule or lexicon window are loaded into Tree Viewer, this action is registered in the Active Windows list. The title of the rule or lexicon window appears in the list.

EXAMPLE: If you save a new set of rules as *Example1.rule*, or load *Example1.rule* using the Load button, the active windows list changes to look like this:



Since the user may have multiple rule and lexicon windows open at a given time, the Active Windows list is helpful for keeping track of which rules or lexical items are currently being used by Syntactica to build trees.

Updating

TreeViewer is not automatically updated on changes made to an active window. This means:

- If you save some rules or lexical items as a file and then make subsequent changes, TreeViewer will not know about those changes until you either save the file again (in which case the changes are written to the file) or click the Load button (in which case the changes are not written to the file).

- If you load a set of rules using the Load button and make subsequent changes, TreeViewer will not be updated until you either save the file or click the Load button again.

Closing

When you close an active rule or lexicon window, its title disappears from the active windows list. This indicates that Syntactica no longer has any active window of that type (rule window, lexicon window).

Inputting Sentences & Generating Phrase markers

Syntactica builds trees for input expressions using rules that have been loaded into TreeViewer, or using rules together with a lexicon.

Building Trees with Rules Alone

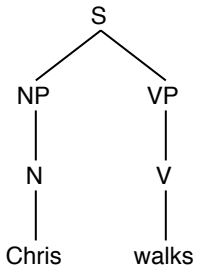
Syntactica can generate phrase markers using rules alone. To do this:

1. Load a set of rules.
2. Enter an input expression in the sentence field.
3. Click the Build Tree button (or `press Return`).

EXAMPLE A: Enter the following rules in a rule window and load them into TreeViewer, either by clicking the Load button or by saving the rules as a file:

S → NP VP
NP → Det N
NP → N
Det → *the*
N → *Chris*
N → *man*
VP → V
V → *walks*

Enter the sentence *Chris walks* in the sentence field, and click the Build Tree button (or `press Return`). The following tree is displayed:



Building Trees with Rules and a Lexicon

Syntactica can also generate phrase markers using rules together with a lexicon. To do this:

1. Load a set of rules.
2. Load a lexicon.
3. Enter an input expression in the sentence field.
4. Click the Build Tree button (or `press Return`).

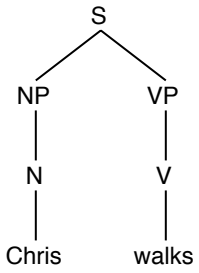
EXAMPLE B: Enter the following rules in a rule window and load them into TreeViewer, either by clicking the Load button or by saving the rules as a file:

S → NP VP
NP → Det N
NP → N
VP → V

Enter the following lexical items in a lexicon window and load them into TreeViewer:

the, Det
Chris, N
man, N
walks, V

Enter the sentence *Chris walks* in the sentence field and click Build Tree (or `press Return`). The same tree is displayed:



Capitalization in the Input

Syntactica observes the following convention in its input:

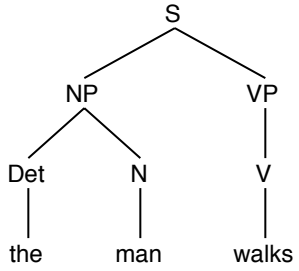
> **Capitalization convention**: The first word in an input sentence may be capitalized, even if the word has not been entered as such in the rules or lexicon.

In all other cases, expressions must be input in exactly the same case form as they were entered in the rules or lexicon.

EXAMPLE: Using the rules in EXAMPLE A above or the rules + lexicon in EXAMPLE B above, input (1) and (2):

1. The man walks
2. the man walks

In both cases Syntactica builds the tree shown below:



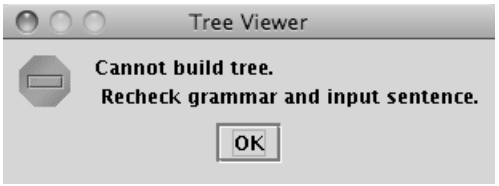
Note that even though input sentence (1) has its first letter capitalized, the tree displays the word in lower case. This matches the way *the* is entered in the rules in EXAMPLE A and in the lexicon in EXAMPLE B.

EXAMPLE: Using the rules in EXAMPLE A or the rules + lexicon in EXAMPLE B, input, enter (3) and (4). The former yields a tree. The latter yields an error message since *Chris* was not entered in lower case in the rules in EXAMPLE A or the lexicon in EXAMPLE B:

3. Chris walks
4. chris walks

The Error Panel

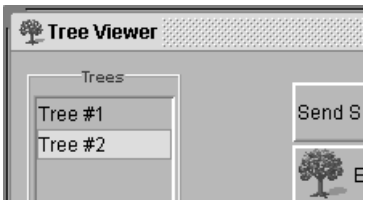
The user can enter any expression as input. However Syntactica will only succeed in building a tree if the input expression is within the scope of the rules and lexicon that have been loaded. If the rules and lexicon do not yield a tree for the input, Syntactica returns an error message:



Viewing Alternative Structures

A given input expression may have several different possible trees under a given grammar. In this circumstance, Syntactica builds all possible trees and allows the user to view them sequentially.

If more than one phrase marker is available for a given input, TreeViewer displays one structure and indicates the presence of others in the Tree List. For example, if two structures are available, the Tree List will look like this:



Tree #2 is highlighted, indicating that it is the tree currently shown in the Tree Display. Other structures can be viewed simply by clicking on their entries in the Tree List.

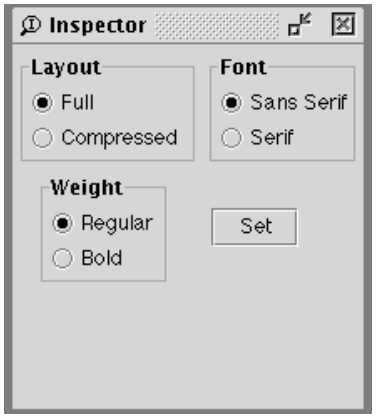
USER NOTE: Clicking back and forth between entries in the tree list will allow you to rapidly inspect structural differences in phrase markers.

Changing Tree Display Format

Trees can be displayed in TreeViewer in several formats.

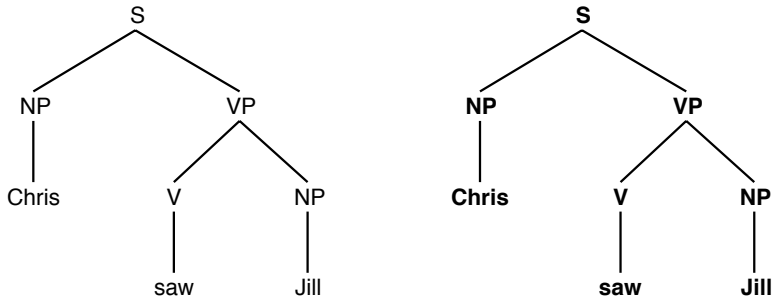
- Nodes and words can be displayed in **sans serif** (Helvetica) or **serif** (Times) **typeface**, and in **regular** or **bold weight**.
- Trees can be displayed in a **full** or **compressed layout**.

To set display parameters, make TreeViewer the key window and select **Options > Inspector**. The TreeViewer Inspector panel appears:

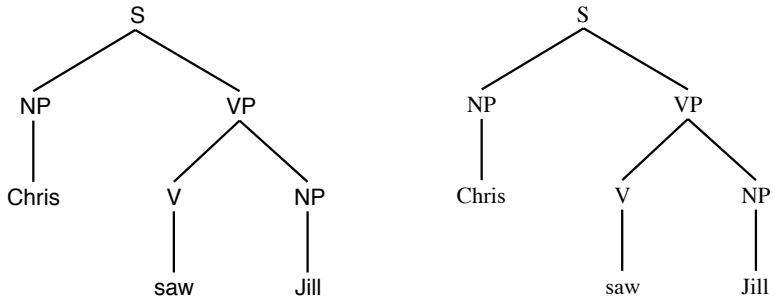


SHORTCUT! Open the Inspector panel with the Button bar:

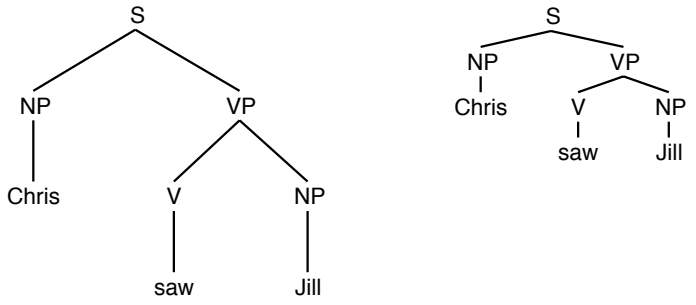
Click the appropriate radio buttons and click Set. The tree redisplays in the selected format. Here are trees in regular vs. bold weight:



Here are trees in sans serif vs. serif font:



And here are trees in full vs. compressed layout:



Displaying Triangles in Trees

Syntactica can display trees containing triangles. To generate a node X dominating a triangle over items A B C, enter a rule of the form:

X ⟶ _A_B_C_

Then use the expression "_A_B_C_" as part of the input for which Syntactic builds a tree.

EXAMPLE: The following rules generate the tree shown below them for the input sentence *_the_man_ gave Fido _to_Jill_*.

S ⟶ NP VP

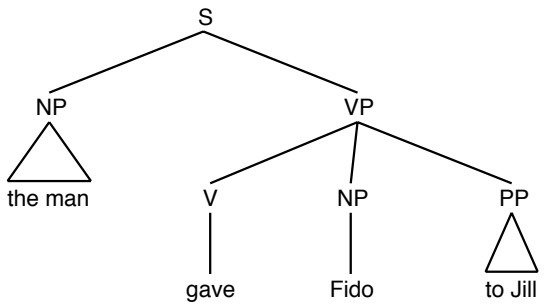
VP ⟶ V NP PP

V ⟶ gave

NP ⟶ _the_man_

NP ⟶ Fido

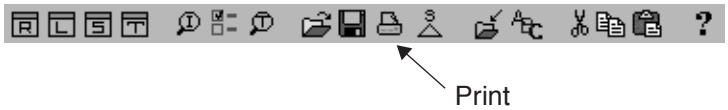
PP ⟶ _to_Jill_



Printing Trees

To print the tree currently shown in the Tree Display, make TreeViewer the key window and select **File > Print**.

SHORTCUT! Click the Print button in the Button bar:



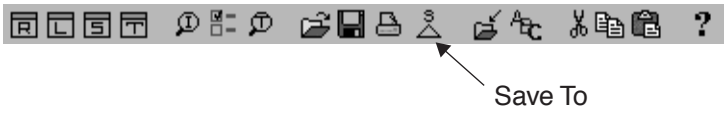
USER NOTE: Trees are printed in the size and format (font, weight and layout) shown in the Tree Display.

Saving Trees as EPS Images

Trees can be saved as individual Encapsulated Postscript (EPS) images for use in homeworks, papers and presentations. To do so:

1. Choose **File > Save To** from the main menu
2. Save the tree as an EPS in the desired location

SHORTCUT! Click the Save To icon in the Button bar:



Syntactica tree images can be opened and printed out in applications that support EPS, such as Adobe Illustrator® or Adobe Acrobat®.

USER NOTE: EPS tree images are saved in the size and format parameters in which they are displayed in TreeViewer.

The PF Display

The PF display gives a surface expression corresponding to the tree in the Tree Display.

The expression in the PF display will usually correspond directly to the terminal string for the tree. The two will diverge, however, when the tree contains elements that Syntactica knows to be inaudible - specifically:

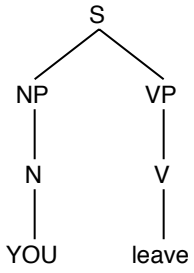
- Lexical items that have been declared as inaudible by the user,
- Empty elements generated by operations like movement and deletion.

For more on the latter, see *Chapter 8 Transformations*.

EXAMPLE: Load the rules and lexicon shown below into TreeViewer, enter the sentence *YOU leave* as input and build its tree:

S → NP VP
NP → N
VP → V
YOU, N, inaudible
leave, V

Syntactica generates a phrase marker in which the inaudible pronoun *YOU* appears:



However, the PF display shows only the verb *leave*. Because *YOU* is declared as inaudible, it does not appear in the PF display:

If a phonetic form is too long to appear in the PF display, you can view the hidden portion by simply clicking in the PF display and moving to the the right with the cursor keys.

Sending Sentences and Trees

The upper portion of the TreeViewer window contains a Send S button and a Send T button. These buttons are used to collect sentence and tree information from TreeViewer into files.

Sending Sentences

The Send S button sends the sentence (or other expression) currently displayed in the sentence field of TreeViewer

from: TreeViewer

to: the active sentence window.

The sentence is added to the sentence list of the active sentence window.

Sending Trees

The Send T button sends the phrase marker currently displayed in the tree display of TreeViewer

from: TreeViewer

to: the active tree window.

The phrase marker is added to the tree list of the active tree window. For information on Sentence window and Tree windows, see *Chapter 7 Sentence Files, Tree Files, and Grammars*.

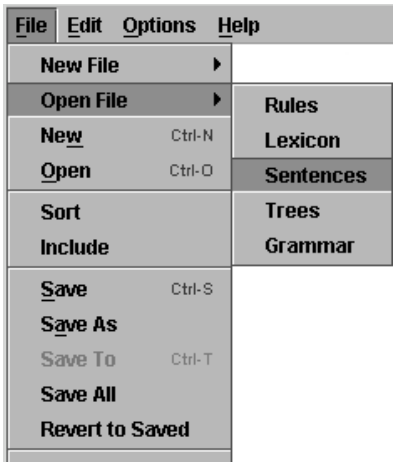
CHAPTER 7

Sentence Files, Tree Files, and Grammars

Sentence Files

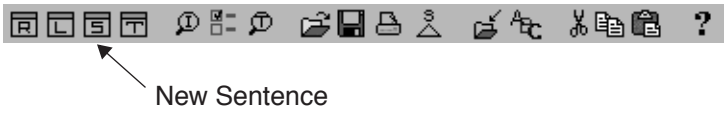
Sentence files consist of a set of sentences, or subsentential expressions. Sentence files have the extension *.sent*.

Sentence files are created, displayed and edited in sentence windows. A sentence window appears whenever you open an existing rule file, or create a new one. To open an existing sentence file, choose **File > Open File > Sentences** from the menu.



To create a new sentence file, choose **File > New File > Sentences**.

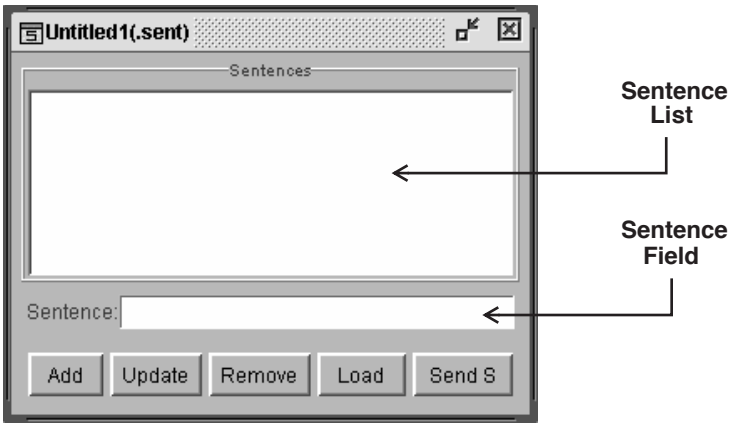
SHORTCUT! Open new sentence windows with the Button bar:



The Parts of a Sentence Window

A sentence window consist of two major parts:

- a sentence list, which displays the expressions in a file,
- a sentence field, for entering expressions.



Under the sentence field are buttons allowing the user to add expressions to the Sentence List, update them, or remove them. In addition there is:

- a Load button, for making a sentence window the active sentence window in TreeViewer,
- a Send S button, for sending an expression from the active sentence window to TreeViewer.

Whenever you select a sentence or other expression in the sentence list, that expression is highlighted in the list and displayed in the sentence field.

Adding, Editing, Saving and Printing Sentences

Sentences (or other expressions) that have been entered in the sentence field can be added to the sentence list, edited (if necessary), saved as a file, or printed.

Adding Sentences to the Sentence List

To add an expression to the sentence list:

1. Enter the expression in the sentence field.
2. Click the Add button.

Editing Sentences in the Sentence List

To edit an expression in the sentence list:

1. Select the sentence in the list.
2. Make changes in the sentence field with the usual text functions.
3. Click the Update button.

Deleting Sentences from the Sentence List

To delete a sentence from the sentence list:

1. Select the sentence in the list.
2. Click the Remove button.

Saving Sentences

To save a list of sentences as a file:

1. Make the sentence window the key window.
2. Choose Save from the main menu (or `enter control s`).

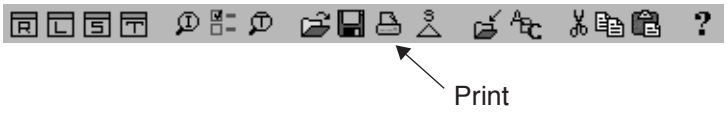
If the file is a new one, the Save panel appears asking you to name the file and give it a location.

Printing Sentences

To print a list of sentences:

1. Make the sentence window the key window.
2. Select Print from the Main menu (or `enter control p`).

SHORTCUT! Click the Print button in the Button bar:



EXAMPLE: Select **File > New File > Sentences**. A new sentence window opens (entitled *Untitled*). Click in the sentence field, type the sentence *John walks*, and click the Add button. The expression in the field now appears as the first line in the sentence list. The window *UNTITLED* should look like this:

Practice by adding the following sentences to the list:

Chris admires Jill

Chris gives Fido to Jill

Jill glarfs

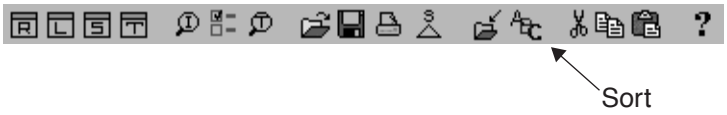
Click on the sentence *Jill glarfs* in the sentence list. Click the Remove button. *Jill glarfs* disappears from the sentence list. Click on the sentence *John walks*. Change *John* to *Chris* in the sentence field (e.g., by dragging across it and typing *Chris*). Click Update. *John walks* changes to *Chris walks* in the sentence list. Make the sentence window the key window and select **File > Save** (or `enter command s`). Save the sentences just entered as *MySentences*.

Sorting Sentences

Lists of sentences can be sorted alphabetically. To do so:

1. Make the window to be sorted the key window.
2. Choose **Edit > Sort** from the application menu.

SHORTCUT! Click the Sort icon in the Button bar:

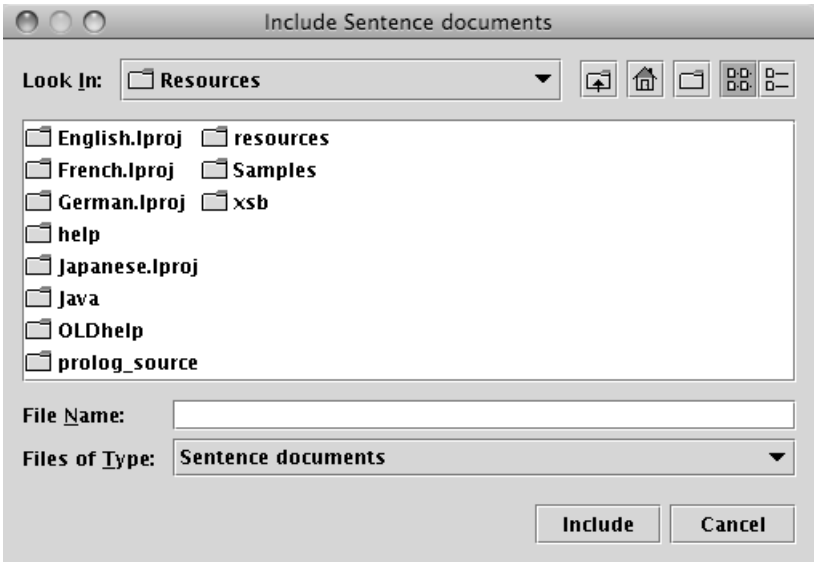


Sorting alters the contents of a window, just like adding or deleting sentences, so remember to save after sorting!

Including Sentences from Other Sentence Files

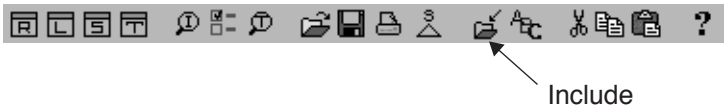
Sentences from other sentence files can be added to those in any open sentence window using the Include command. To do so:

1. Make the window to be added to the key window by clicking on it.
2. Choose **Files > Include** from the Main menu (or `enter control i`). A panel appears asking you to select the sentence file whose content is to be included.



3. Select the file of sentences to be included.
4. Click Include (or `press Return`).

SHORTCUT! Click the Include icon in the Button bar:



Loading Sentences into TreeViewer

A sentence window is loaded into TreeViewer whenever:

- The sentence window is saved as a file, or
- The Load button in the sentence window is clicked

Loading a sentence window causes its title to be displayed in TreeViewer's active windows list. This signifies that a link has been established between the sentence window and TreeViewer. This link is exploited with the Send S button.

The Send S Button

Clicking the Send S button of the active sentence window sends the expression currently displayed in the sentence field

from: the active sentence window
to: the sentence field in TreeViewer.

This facility makes it easy to test a given file of sentences (or other expressions) against a set of rules (or rules and lexicon) that have been loaded. For each sentence, the user:

- Selects a sentence
- Clicks the Send S button (sending it to TreeViewer)
- Clicks the Build Tree button

Send S makes it unnecessary to copy and paste expressions into Tree-Viewer's sentence field.

Closing

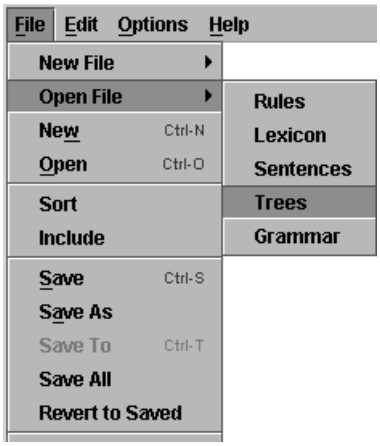
When you close an active sentence window, its title disappears from the active windows list. This indicates that Syntactica no longer has any active sentence window

Tree Files

A tree file contains a set of trees. Tree Files have the extension *.tree*.

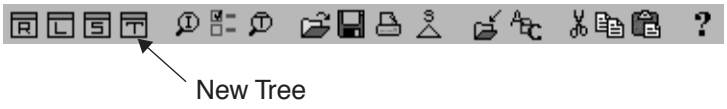
Tree files are created and displayed in tree windows. A tree window appears whenever you open an existing tree file, or create a new one. To

open an existing tree file, choose **File > Open File > Trees** from the menu.



To create a new tree file, choose **File > New File > Trees**.

SHORTCUT! Open new tree windows with the Button bar:

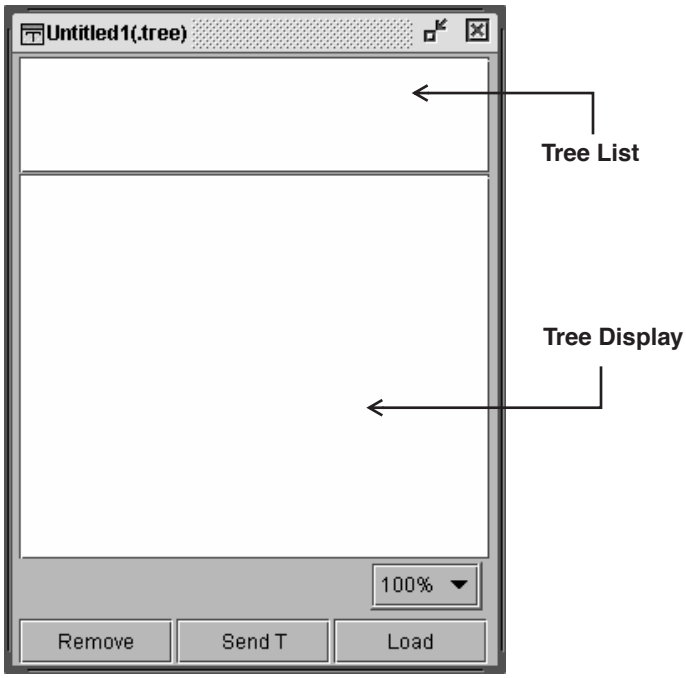


The Parts of a Tree Window

A tree window consist of:

- a Tree List, which lists the trees in a file by their terminal strings
- a Tree Display, for exhibiting trees

Under the Tree Display are buttons allowing the user to remove trees, to resize them, to send them to TreeViewer, or to load the window, making it the active tree window.



Whenever you select a tree in the Tree List, it is highlighted in the list and displayed in the Tree Display.

Adding, Editing, Saving and Printing Trees

Trees generated in TreeViewer are collected in tree windows, and saved as files.

Adding Trees to Tree Window

To add a tree to a tree window:

1. Load the tree window into TreeViewer (see below), making it the active tree window.
2. Build the desired tree in TreeViewer.
3. Click the Send T button in TreeViewer.

Sending a tree from TreeViewer adds it to the tree list of the active tree window.

Deleting Trees from Tree List

To delete a tree from the tree list:

1. Select the tree in the list.
2. Click the Remove button.

Saving Trees in Tree Window as a File

To save a tree window as a file:

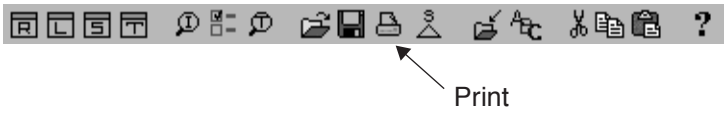
1. Make the tree window the key window by clicking on it.
2. Choose **File > Save** from the main menu (or `enter control s`).

If the file is a new one, the Save Tree Documents panel appears asking you to name the file and give it a location.

Printing Trees

To print the tree currently shown in the Tree Display of a tree window, make the latter the key window and select **File > Print**.

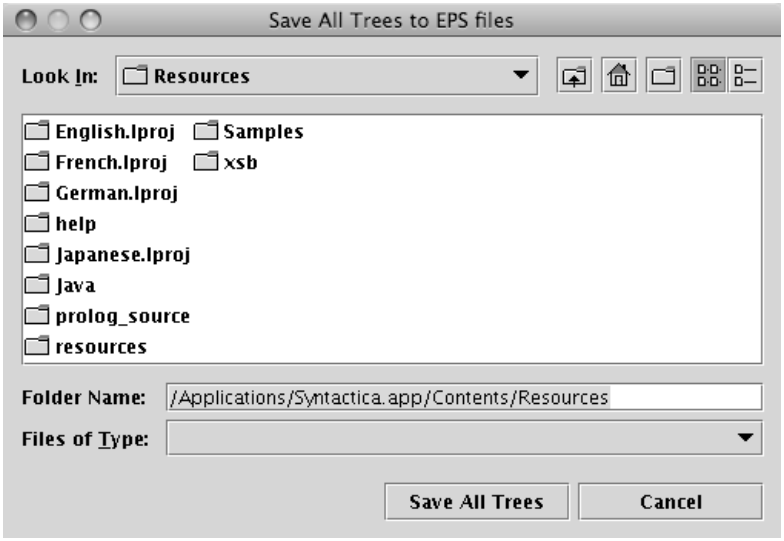
SHORTCUT! Click the Print button in the Button bar:



Saving a Folder of Tree Images

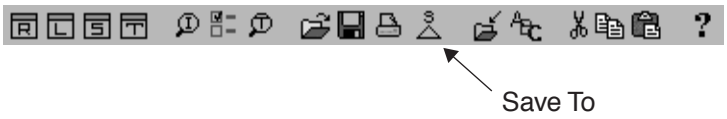
The trees in a tree window can be saved as a folder of Encapsulated Postscript (EPS) images for use in homeworks, papers and presentations. To do so:

1. Choose **File > Save To** from the main menu. A panel appears asking you to locate the folder where the tree images are to be saved, or to create a new folder by appending its name to the directory path indicated:



2. Save the trees in a folder in the desired location

SHORTCUT! Click the Save To icon in the Button bar:



Syntactica tree images can be opened and printed out in applications that support EPS, such as Adobe Illustrator® or Adobe Acrobat®.

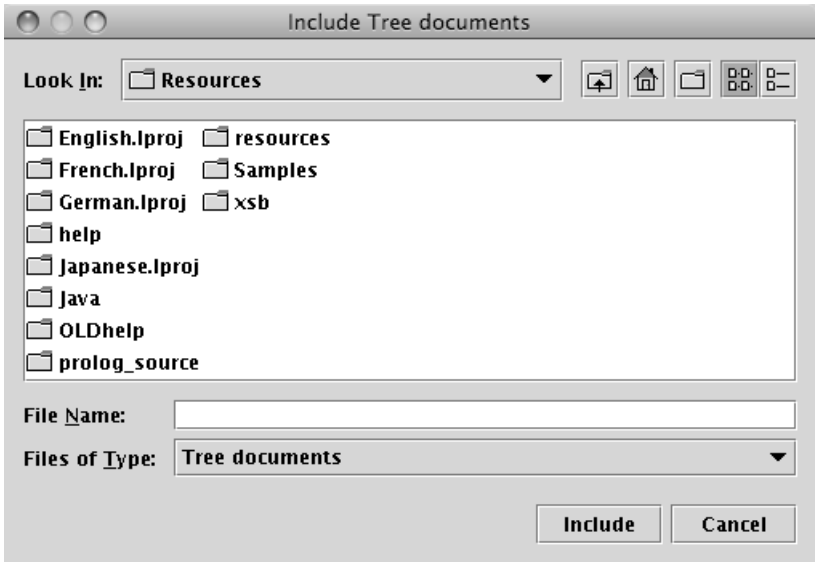
USER NOTE: Trees are displayed in tree windows in the format currently selected for TreeViewer. Trees in a tree window are also saved in the format currently displayed in TreeViewer. Therefore, to save the trees of a tree window in a given format, make the appropriate formatting selections in TreeViewer before saving.

Including Trees From Other Tree Files

Trees from other tree files can be added to those of any open tree window. using the Include command. To do so:

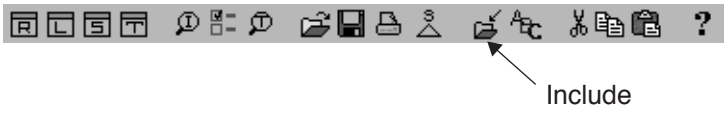
1. Make the window to be added to the key window by clicking on it.

2. Choose **Files > Include** from the Main menu. A panel appears asking you to select the tree file whose contents are to be included.



3. Select the file of trees to be included.
4. Click Include (or `press Return`).

SHORTCUT! Click the Include icon in the Button bar:



Loading Trees into TreeViewer

A tree window is loaded into TreeViewer whenever:

- The tree window is saved as a file, or
- The Load button in the tree window is clicked

Loading a tree window causes the title of that window to be displayed in TreeViewer's active windows list and creates a link between the tree window and TreeViewer. This link is exploited with the Send T button.

The Send T Button

Clicking the Send T button of the active tree window sends the tree currently exhibited in the tree display

from: the active tree window
to: the tree display of TreeViewer.

This facility makes it easy for a user to experiment with syntactic operations on an existing file of trees. In the Tree window the user simply:

1. Selects a tree in the tree list.
2. Clicks the Send T button (sending it to TreeViewer).
3. Performs operations using the Transforms panel.

Send T makes it unnecessary to generate each tree separately in the TreeViewer tree display, if the trees have already been saved in a file.

Closing

When you close an active tree window, its title disappears from the active windows list. This indicates that Syntactica no longer has any active tree window.

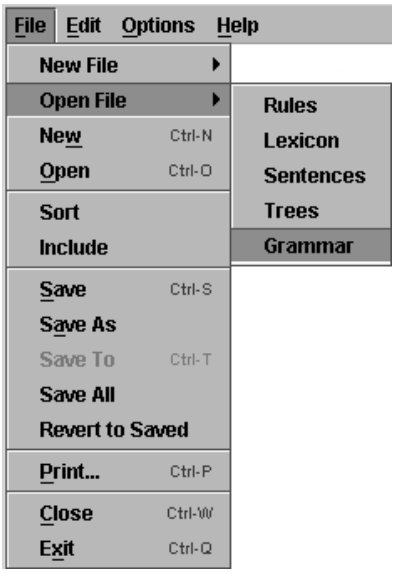
Grammars

Syntactica allows rule and lexicon files to be saved together as grammars. Files in a grammar have the extension *.rgr* and *.lgr*.

What is a Grammar?

A grammar is a pair consisting of a single rule file and a single lexicon file. Grammars have no special display window of their own. When you open a grammar by choosing **File > Open File > Grammar** from the menu, separate rule and lexicon windows are opened to display the two components of the grammar. These rules and lexicon are also loaded into TreeViewer, and their titles are displayed in the active windows list.

Grammars are useful for keeping together rule sets and lexicons that have been created together. Rather than storing them as separate entities, whose



connection may be forgotten, the user may elect to store them as a single unit that can be loaded, modified and saved as single unit.

Creating a Grammar

Grammars are created by saving the rules and lexicon together. To create a grammar, first load rules and a lexicon into TreeViewer. Then make the TreeViewer window the key window and select **File > Save** (or `enter control s`). The Save Grammar Documents panel appears asking you to specify a name and location for the grammar.

When you save a grammar under a name - for example, *File* - Syntactica makes a copy of the rule and lexicon files. The component rule file is saved as *File.rgr*. The component lexicon is saved as *File.lgr*.

Editing, Saving and Loading Grammars

Grammars are not a separate file-type in Syntactica. Editing, saving and loading them is similar to what occurs with rules and lexicons.

Editing a Grammar

To edit a grammar, simply edit its component rule and lexicon files as you normally would.

Saving an Existing Grammar

To save changes to a grammar that has already been created, simply save changes to its component rule and lexicon files in the normal way.

Loading a Grammar

A grammar is loaded whenever it is opened. A grammar can also be loaded by loading its component rule and lexicon files separately in the usual way.

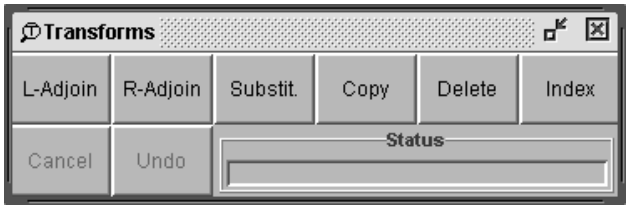
CHAPTER 8

Transformations

Trees generated by Syntactica can be transformed by various syntactic operations, including movement, copying, deletion, and indexing. These operations are performed using the Transforms panel.

The Transforms Panel

To open the Transforms panel by choose **Options > Transforms** from the menu (or `enter control T`):



In the upper row are six buttons used to perform syntactic operations. In the lower row is a Cancel button, which aborts an operation in progress, an Undo button, which undoes a completed operation, and a status field, which reports where the user is in an operation.

SHORTCUT! Click the Transforms icon in the Button bar:

Types of Operations

To transform trees in Syntactica, the user selects nodes in a tree by clicking on them, and clicks an operation button in the Transforms panel. Syntactica performs **unary operations** and **binary operations**.

Unary Operations

In a unary operation, the user selects a node (or set of nodes) in a tree, and clicks an operation button in the Transforms panel. The selected nodes are the **target nodes**. The unary operations are:

- deletion,
- indexing.

Binary Operations

In a binary operation, the user selects a node (or set of nodes) in the tree, clicks an operation button in the Transforms panel, then selects another node in the tree. The nodes initially selected are the **source nodes**. The second node selected is the **target node**. The binary operations are:

- copying,
- left-adjunction,
- right-adjunction,
- substitution.

The last three are also referred to as **movement operations**.

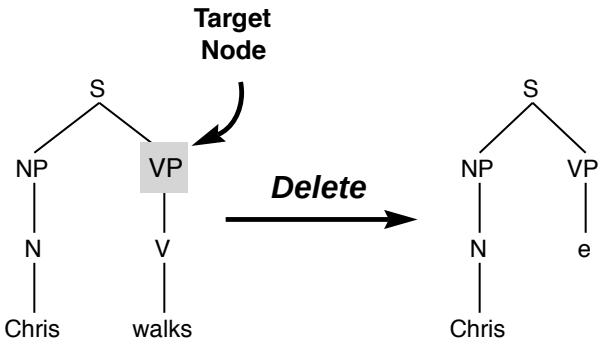
Deletion

To perform a deletion:

1. Select a node in a tree.
2. Click the Delete button in the Transforms panel.

When the target node is selected, it is highlighted in the tree. The deletion operation elides all of the children of the selected node and replaces them with the symbol used to denote the empty string.

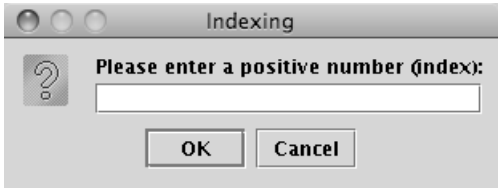
EXAMPLE: If the highlighted VP node is deleted, the result is as shown:



Indexing

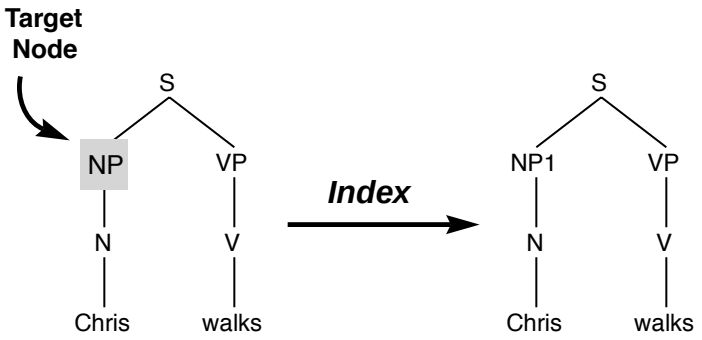
To perform an indexing:

1. Select a node in a tree.
2. Click the Index button in the Transforms panel. The Indexing panel appears:



3. Enter any positive integer in the index field and click OK (or `press Return`).

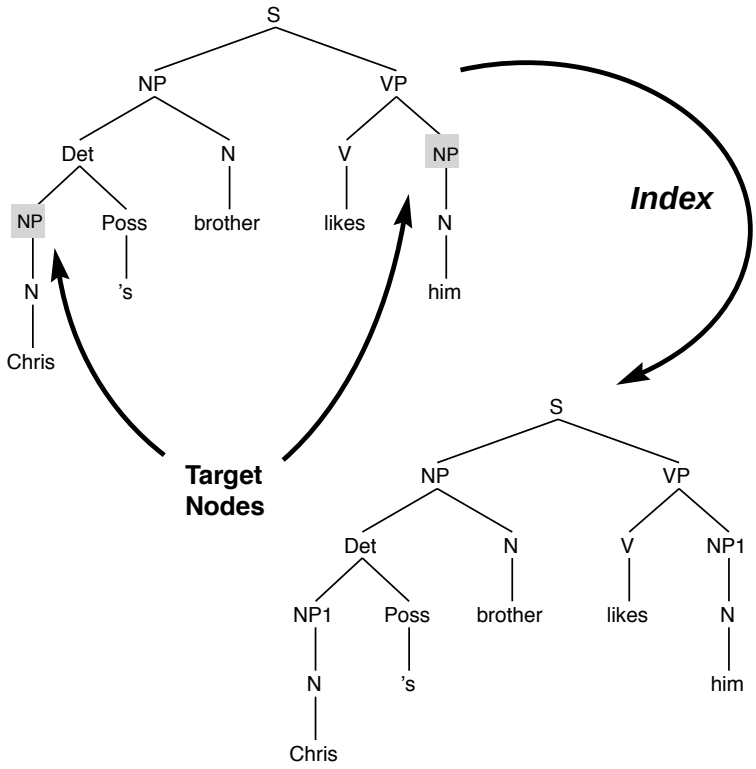
EXAMPLE: If the highlighted NP node is indexed with the integer 1, the result is as shown:



Multiple Selection

Syntactica allows multiple target nodes to be selected via the usual Ctrl-click option. The user selects one node and, while holding the Control key down, selects another. In the case of indexing, this allows coindexation of a number of phrases simultaneously.

EXAMPLE: If the two highlighted NP nodes are simultaneously indexed with the integer 1, the result is as shown:



Note that the same result would have been obtained by selecting the individual NP nodes and indexing them with 1 in two separate operations.

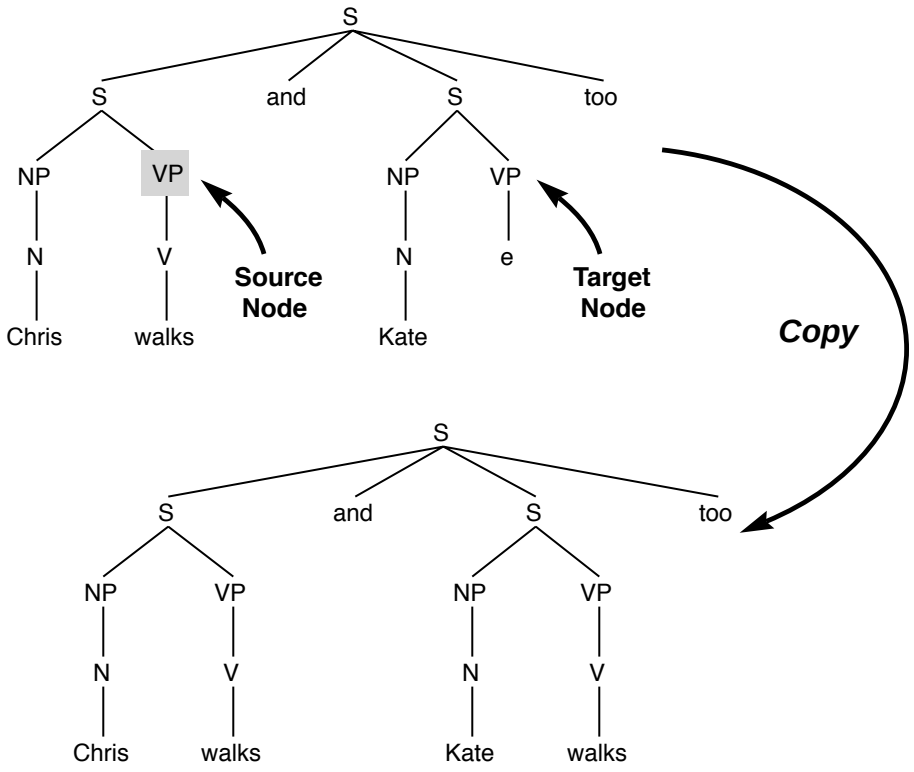
Copying

To perform a copying:

1. Select a source node in a tree.
2. Click the Copy button in the Transforms panel.
3. Select a target node.

When the source node is selected, it is highlighted in the tree. When the target node is selected, Syntactica replaces the subtree dominated by the target node with the subtree dominated by the source node.

EXAMPLE: If the highlighted VP node is copied into the empty VP node, the result is as shown:

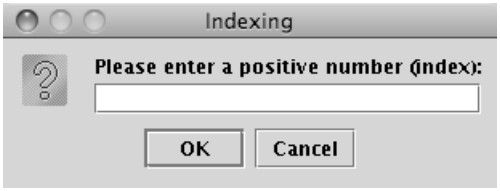


The copying operation respects two special constraints: The Identical Category Constraint and the Empty Target Node Constraint. For more on this, see *Chapter 9 Constraints on Transformations*

Left-Adjunction

To perform a left-adjunction operation:

1. Select a source node in a tree.
2. Click the L-Adjoin button in the Transforms panel.
3. Select a target node in the tree. The indexing panel appears.:

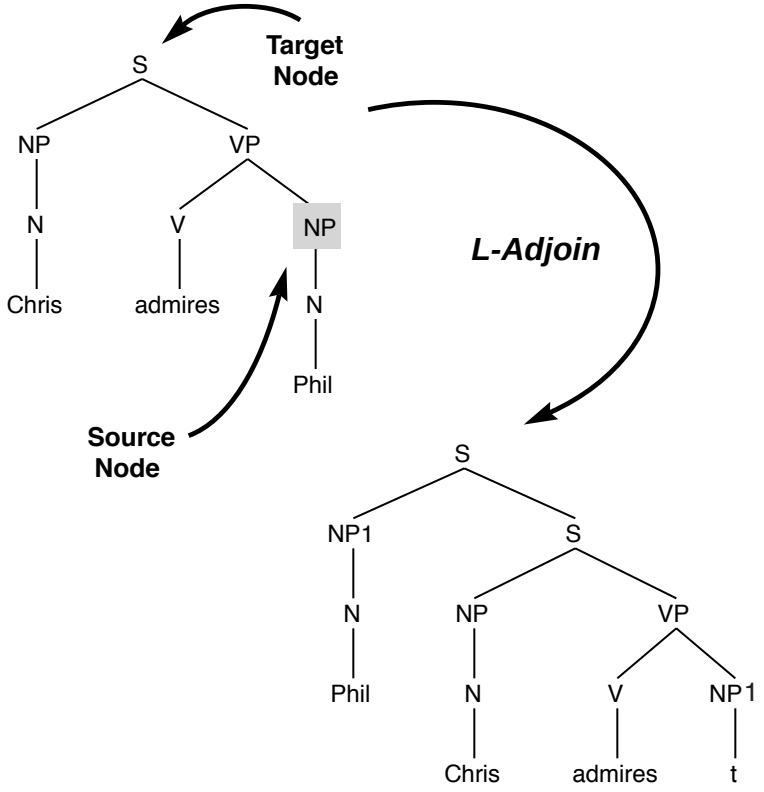


4. Enter any positive integer in the index field and click OK (or `press Return`).

In a left-adjunction, the source node corresponds to the node that is moved and adjoined. The target node corresponds to the node that is adjoined-to. The left-adjunction operation does the following things:

- adjoins the source node on the left of the target node,
- leaves a trace in the original position of the source node,
- coindexes the trace and the moved phrase using the index that the user has entered.

EXAMPLE: If the highlighted NP node is moved and left-adjoined to the S node (and the index 1 is entered), the result is as shown:

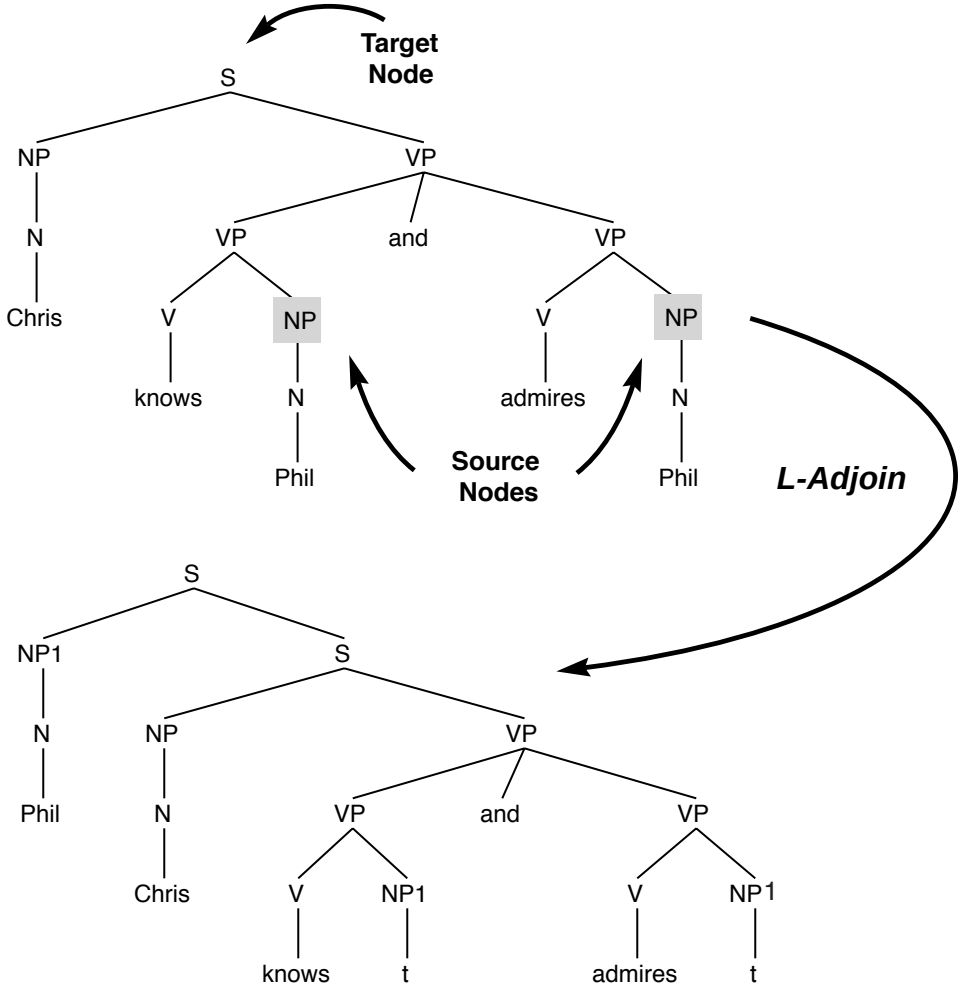


Syntactica enforces certain general constraints on movement operations, including left-adjunction. For more information, see *Chapter 9 Constraints on Transformations*.

Multiple Selection

Multiple source node selection (via the usual Ctrl-click option) allows the user to move a number of phrases simultaneously.

EXAMPLE: If the two highlighted NP nodes are simultaneously moved from the conjoined VPs and left-adjoined to the S node, the result is as shown:



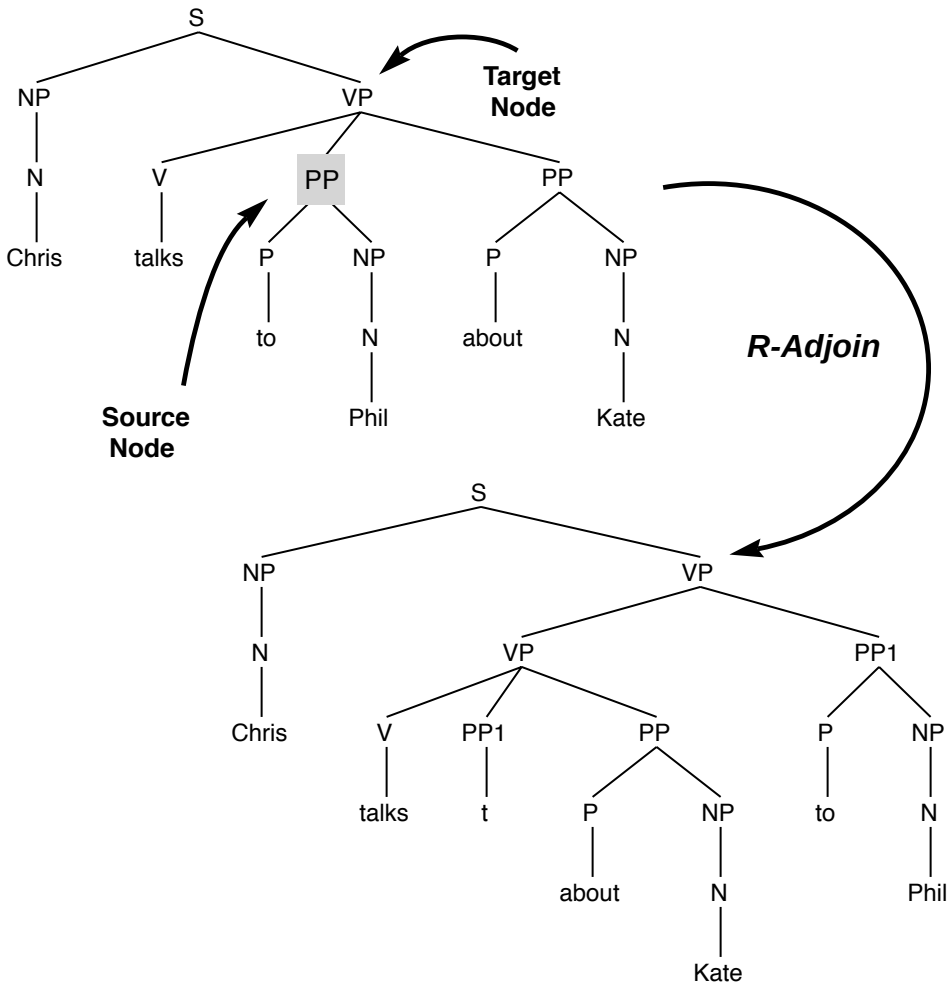
Movement of this kind is called **across-the-board (ATB) movement**. A phrase is moved simultaneously from two or more source positions to a single target position.

Syntactica enforces certain constraints on ATB movements. For more information, see *Chapter 9 Constraints on Transformations*.

Right-Adjunction

Right-adjunction is performed identically to left-adjunction, obeys the same constraints and has essentially the same effects, except that it adjoins the source node on the right of the target node.

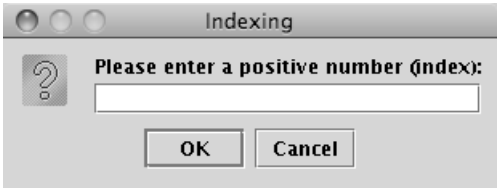
EXAMPLE: If the highlighted PP node is moved and right-adjoined to the VP node, the result is as shown



Substitution

To perform a substitution operation:

1. Select a source node in a tree.
2. Click the Substitute button in the Transforms panel.
3. Select a target node in the tree. The indexing panel appears.:

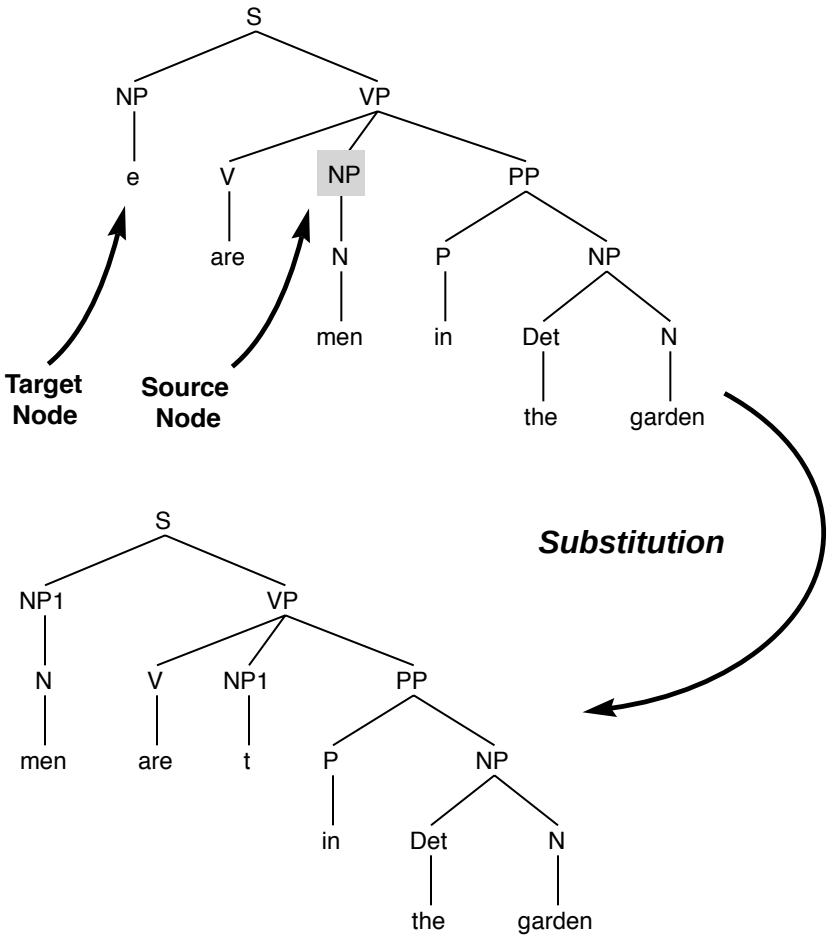


4. Enter any positive integer in the index field and click OK (or `press Return`)

The substitution operation does the following things:

- substitutes the source node for the target node,
- leaves a trace in the original position of the source node,
- coindexes the trace and the moved phrase using the index that the user has entered.

EXAMPLE: If the highlighted NP node is moved and substituted for the empty NP node in subject position, the result is as shown:

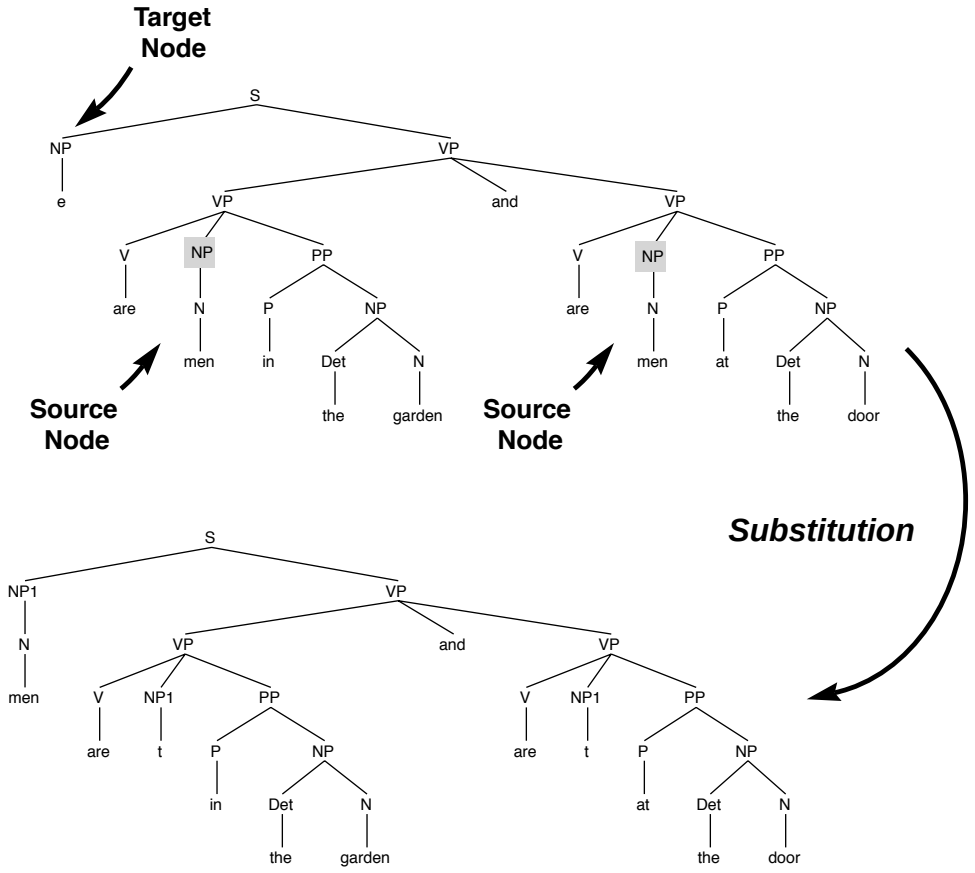


Syntactica enforces certain general constraints on movement operations, including substitution. Syntactica also enforces specific constraints on substitutions. For more information, see *Chapter 9 Constraints on Transformations*.

Multiple Selection

Multiple source node selection using Ctrl-click allows the user to substitute a number of phrases simultaneously.

EXAMPLE: If the two highlighted NP nodes are simultaneously moved and substituted for the empty NP node in subject position, the result is as shown:

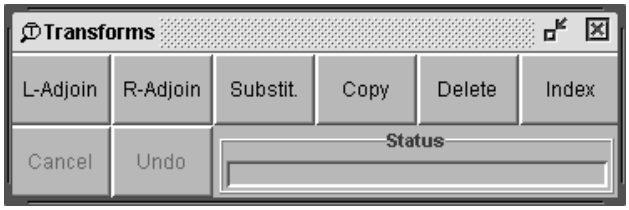


Movement of this kind is called **across-the-board (ATB) movement**. A phrase is moved simultaneously from two or more source positions to a single target position.

Syntactica enforces certain constraints on ATB movements. For more information, see *Chapter 9 Constraints on Transformations*.

Canceling and Undoing Operations

Operations performed by Syntactica can be canceled or undone using the Transforms panel:

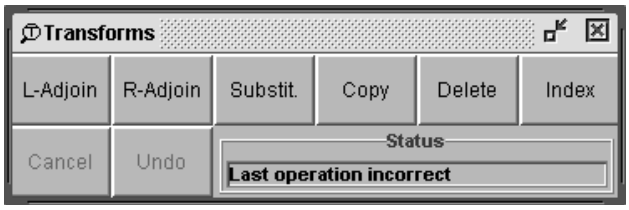


The Cancel button is used to stop an operation that is in progress and has not yet been completed. The Undo button is used to undo an operation that has already been completed. Undo undoes the last operation performed.

The status field informs the user of whether an operation is in progress. Typically:

- An empty status field means that no operation is in progress. Only Undo can be used.
- A filled status field means that an operation is in progress. Either Cancel or Undo can be used.

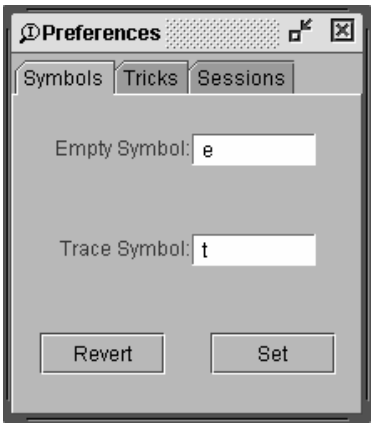
An exception is when an incorrect or illegal operation has been attempted. In this case, the message *Last Operation Incorrect* appears in the status field even though no operation is in progress:



Setting the Empty Symbol & Trace Symbol

The user can set the symbols that Syntactica employs to denote the empty string and the trace of movement using the Preferences panel.

Select **Options > Preferences** from the menu. The following panel appears:



The default symbol for the empty string is *e*. The default symbol for trace is *t*. To change these defaults, edit the relevant fields and click Save. Any contiguous string of characters that can be typed from the keyboard can be used as the empty symbol or the trace symbol.

USER NOTE: Syntactica always requires an empty symbol to be declared. The trace symbol may be left undeclared. If no trace symbol is declared, Syntactica uses the empty symbol as its trace symbol.

CHAPTER 9

Constraints on Transformations

Some transforms performed by Syntactica are subject to special constraints. These restrict movements of various kinds and also certain kinds of copyings.

General Constraints on Movement Operations

All movement operations performed by Syntactica (i.e., left-adjunctions, right-adjunctions, and substitutions) respect two constraints:

- the Subjacency Constraint
- the Identical Source Node Constraint on ATB Movement.

The Subjacency Constraint

The Subjacency Constraint restricts the possible distance that may intervene between source node and target node in a movement operation. The constraint is the following:

> **Subjacency**: In any movement operation, the moved item may cross at most one of a designated set of bounding nodes

If Subjacency is violated in a derivation, Syntactica returns an error message:

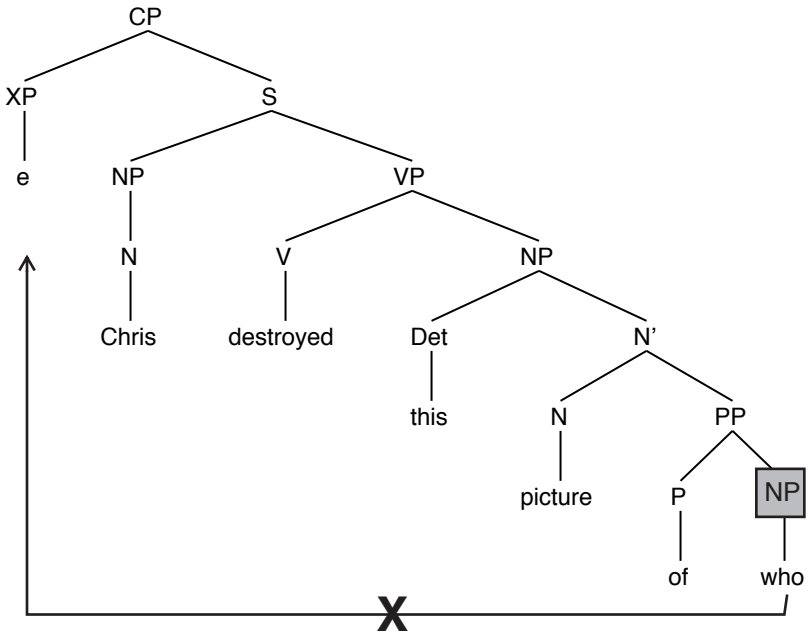
Bounding nodes for Subjacency are set with the Rules Inspector panel. To specify a bounding node, make the rule window the key window and select **Options > Inspector** from the menu. The following panel appears:

Syntactica's default is to set no bounding nodes. Enter the relevant categories in the lower field of the Bounding Nodes area and Press Return. Then click Set. Save or load the rule file. Syntactica will then block any movement that crosses two or more nodes from the declared set of bounding nodes (henceforth BNs).

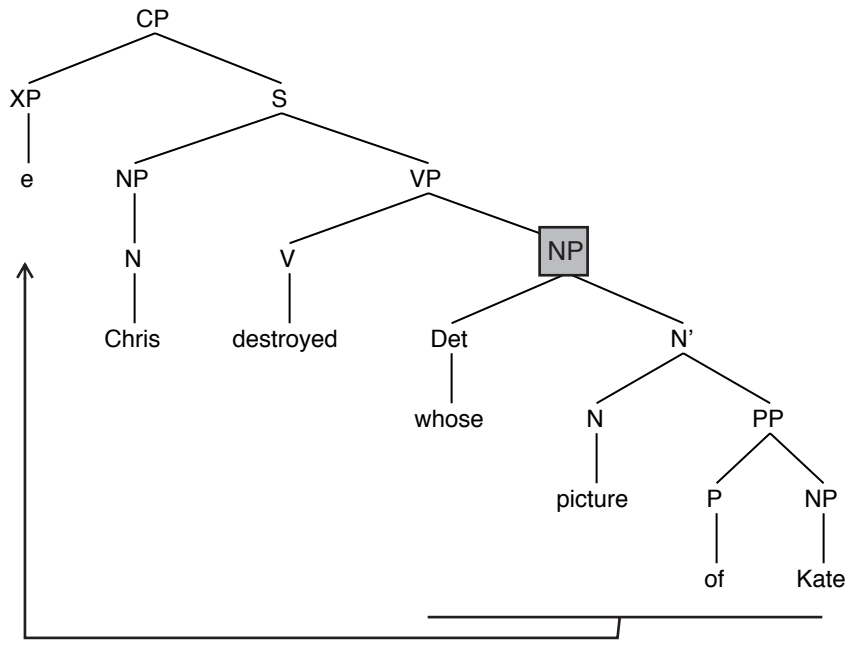
Subjacency with Substitutions

Subjacency constrains both substitutions and adjunctions. But it applies rather differently in the two cases.

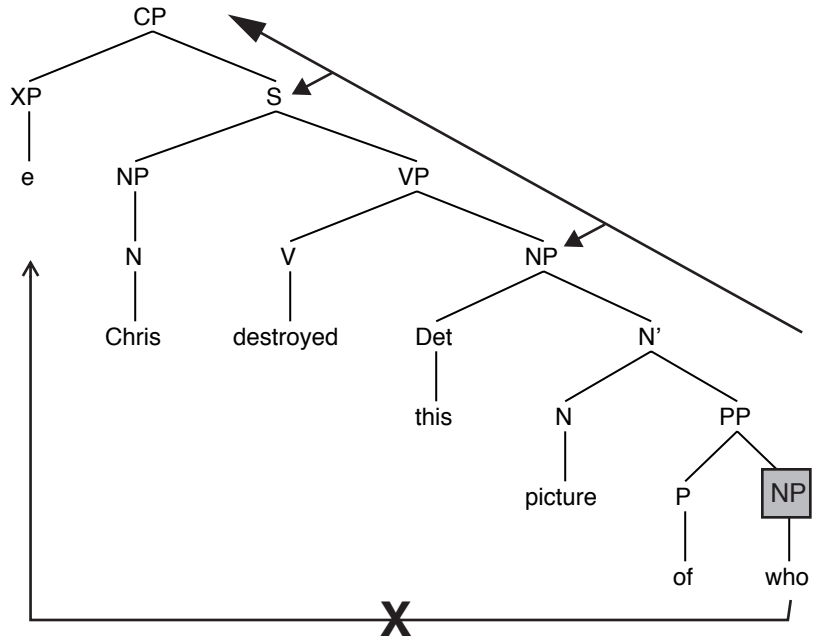
EXAMPLE: With S and NP declared as BNs, the Subjacency Constraint will block the substitution shown below (compare *?*Who did Chris destroy this picture of?*):



By contrast, the following substitution will succeed (compare *Whose picture of Kate did Chris destroy?*):



Subjacency is calculated in these examples by climbing upwards along the spine of the tree, from the source node to the mother of the target node, checking for BNs along the way. Two BNs (NP and S) are encountered in the first tree shown above. This violates Subjacency:



By contrast, only one BN is encountered in the second tree, so no violation of Subjacency occurs.

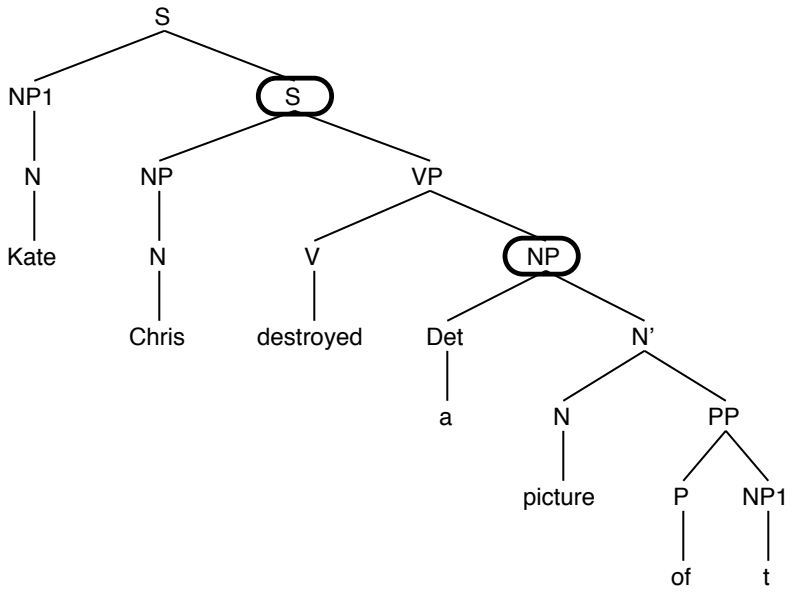
Subjacency with Adjunctions

Adjunction operations attach expressions to nodes, including nodes that may have been declared as BNs. In these circumstances Syntactica calculates as follows:

> **Bounding Nodes for Adjunctions**: Nodes adjoined-to do not count for Subjacency.

In effect, attaching to a node does not count as crossing that node insofar as Subjacency is concerned.

EXAMPLE: With S and NP declared as BNs, the Subjacency Constraint will allow the adjunction shown below, even though two BNs appear to have been crossed:

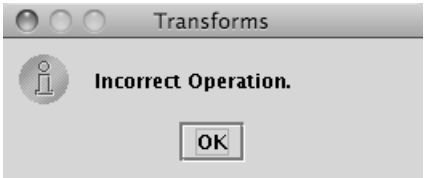


The Identical Source Node Constraint on ATB Movement

Syntactica enforces the following general constraint on movements that involve more than a single source node, i.e., movements that are across-the-board (ATB):

> **Identical Source Node Constraint**: ATB movement is permitted only when the selected source nodes are identical.

Syntactica will allow the user to multiply-select any set of source nodes. However, if those nodes are non-identical, movement fails and Syntactica returns an error message:



Identity of source nodes includes identity of indices that may have been added by Syntactica in the course of performing an operation (indexing, movement, etc.).

USER NOTE: The "Tricks" tab in Preferences contains an "ATB pre-select" radio button. Choosing ATB pre-select permits only the selection of identical source nodes for syntactic operations.

Constraints on Substitutions

Substitution respects two special constraints:

Empty Target Node Constraint: The target node for substitution must be an empty node - a node dominating the empty string - or else a node dominating a designated pleonastic element.

Like-Category Constraint: The target node for substitution must be identical in category to the source node, or else must be of a designated variable category.

When either constraint is violated, Syntactica returns an error message:

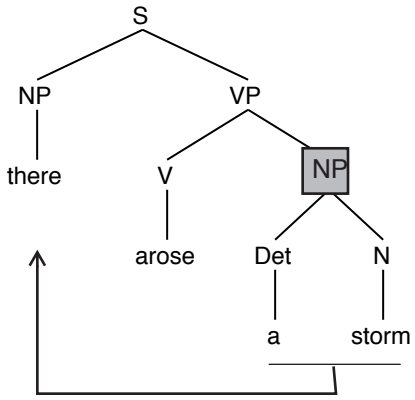
The Empty Target Node Constraint

The Empty Target Node Constraint requires that substitutions preserve information in a derivation. The content of a nonempty node cannot be effaced unless it consists of a specially declared item - a **pleonastic** - whose identity is declared.

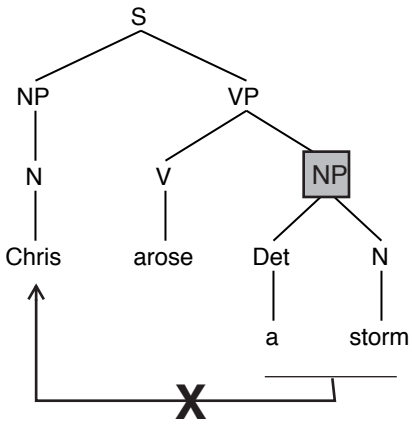
Pleonastic elements are declared with the Lexicon Inspector. To declare a pleonastic, make the lexicon window the key window and select **Options > Inspector**. The following panel appears:

Enter the relevant item in the lower field of the Pleonastic Words area and press Return. Then click Set. Save the lexicon file. Syntactica will now permit substitution to efface the designated element.

EXAMPLE: With *there* declared as pleonastic, the following substitution will succeed:



By contrast, the Empty Node Constraint will block the following substitution:



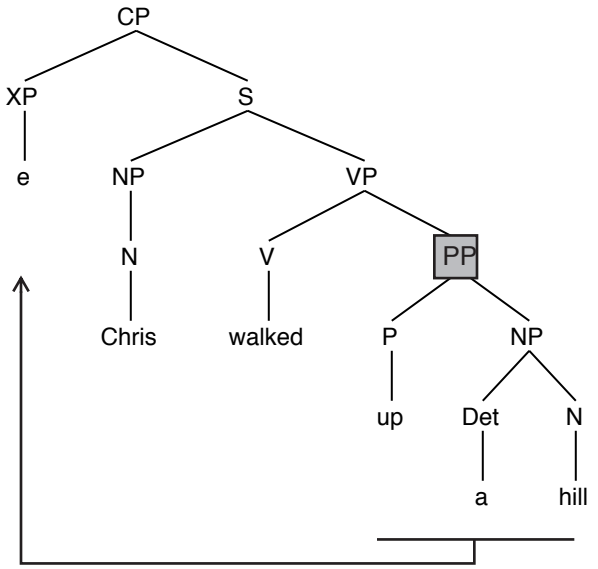
The Like-Category Constraint

The Like-Category Constraint also requires that substitutions preserve information. The category of a node cannot be effaced unless it is a special declared variable category.

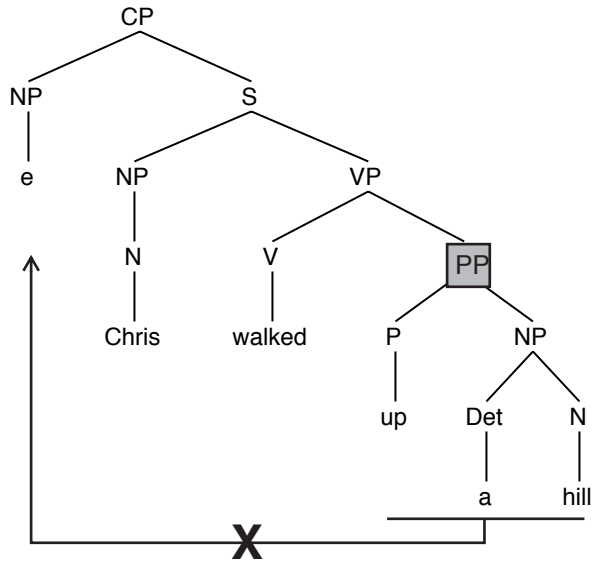
Variable categories are declared using the Rules Inspector. To declare a variable category, make the rule window the key window and select **Options > Inspector** from the menu. The following panel appears:

Syntactica's default variable category is XP. Change this value (if desired) by clicking on “XP” in the upper field of the Variable Category area, editing this element in the lower field, and pressing Return. Then click Set. Save or load the rule file. Syntactica will now permit substitution to cover the designated category.

EXAMPLE: With XP declared as a variable category, the substitution shown below will succeed:

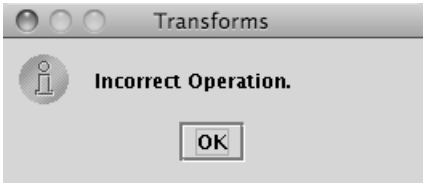


By contrast, the Like-Category Constraint will block the following substitution:

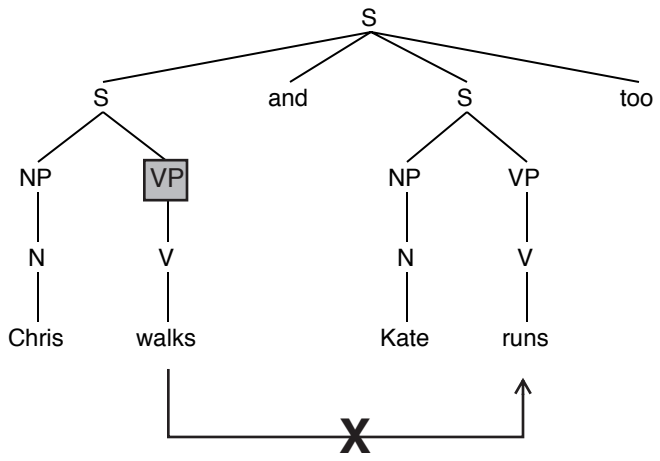


Constraints on Copying

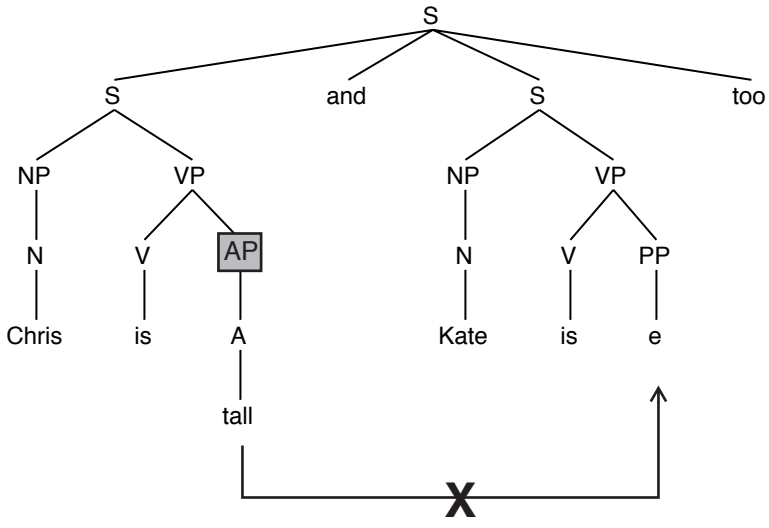
The copying operation respects the Empty Target Node Constraint and the Like-Category Constraint discussed above. When either constraint is violated in a derivation, Syntactica returns an error message:



EXAMPLE: The requirement that the target of copying be empty will block the derivation shown below:



EXAMPLE: The requirement that the target of copying be of identical category will block the derivation shown below:



Constraints on Copying versus Substitution

The empty target and like-category constraints on copying and substitution are similar. The differences are the following:

- Copying over a pleonastic is not permitted, whereas substitution over a pleonastic is permitted.
- Copying into a variable category is not permitted, whereas substitution into a variable category is permitted.

We might summarize by saying that copying obeys the empty target and like-category constraints in a stronger way - one allowing for no exceptions.

CHAPTER 10

Heads, Features, and Complements

Syntactica enforces certain relationships among three types of information:

- headedness in rules,
- features of lexical items,
- features of complements.

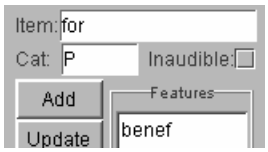
These interact in a complex way in Syntactica. It is useful to distinguish feature-rule interactions and complement-feature-rule interactions.

Feature and Rule Interactions

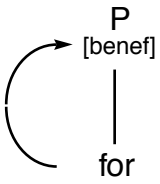
The main principles governing feature-rule relationships are:

1. The features of a lexical item are inherited by the lexical node that dominates it.
2. In a rule, the mother node inherits the features of any category specified as a head.

EXAMPLE: Suppose that *for* is entered in the lexical item template as shown below:



Then according to the first principle, the feature [benef] assigned to *for* is inherited by the P node that dominates it

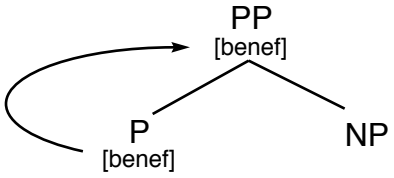


NB: **the features shown here and below are for illustrative purposes only. They are <u>not</u> actually displayed in the Tree Viewer window**.

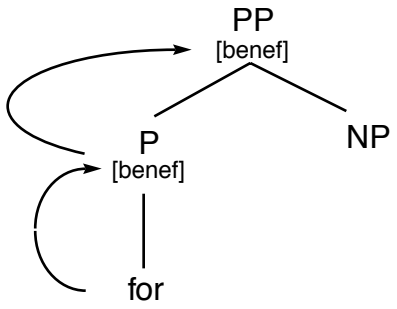
EXAMPLE: Suppose the rule for PPs is specified as follows (where boldface indicates that the category is a head):

PP → **P** NP

Then according to the second principle, any features of P will be inherited by PP. In particular, if the feature [benef] is present on P, it will be inherited by PP:



It follows from principles one and two that if *for* is specified as [benef], and if P is specified as a head in the rule PP → P NP, then the feature [benef] will be inherited by PP:



Complement, Feature and Rule Interactions

Complement-feature-rule relationships are governed by the constraints on features and rules stated above. That is:

1. The features of a lexical item are inherited by the lexical node that dominates it.
2. In a rule, the mother node inherits the features of any category specified as a head.

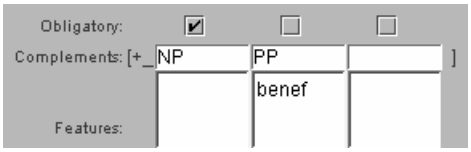
But complements also involve two new principles:

3. Complement information is feature information.
4. The complement features of a lexical node are checked by the mother of that lexical node.

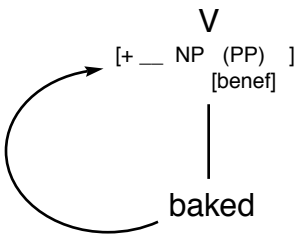
Inheriting Complement Information

The fact that complement information is feature information entails that it may be inherited upwards.

EXAMPLE: Suppose the verb *baked* is entered with its complements specified as shown below:



Recall from *Chapter 5* that the checked box above NP signifies that *baked* must be followed obligatorily by a noun phrase object, and the unchecked box above PP signifies that the NP object of *baked* may be followed, optionally, by a benefactive PP (one containing *for*). The principle that complement information is feature information entails that this complement information of *baked* is inherited by the V node that dominates it:



In this figure, the parentheses around the PP indicate that it is optional, and the [benef] below PP indicates that PP has the feature [benef].

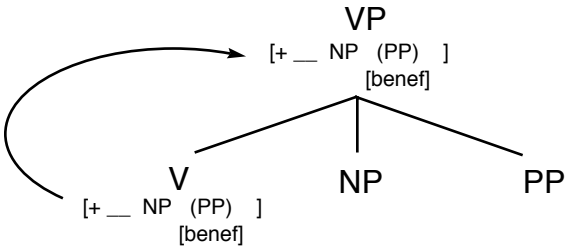
EXAMPLE: Suppose the rule for VPs is specified as follows (where boldface indicates a head):

VP → **V** NP PP

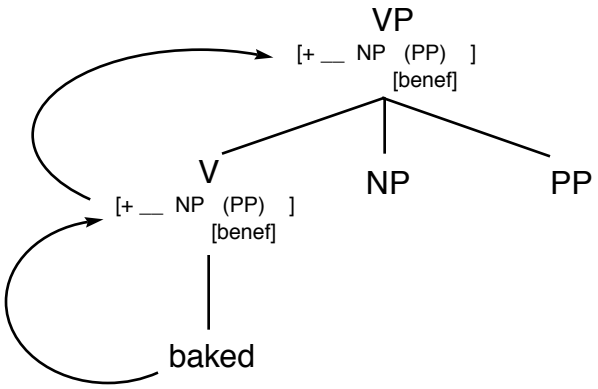
Then according to the second principle, the features of V will be inherited by VP. In particular, if the feature:

[+ __ NP (PP)]
[benef]

is present on V, it will be inherited by VP:



It follows principles one and two that if *baked* is specified as above, and if V is specified as the head of the rule VP → V N PP, then the complement feature will be inherited by VP:



Enforcing Co-occurrence Restrictions

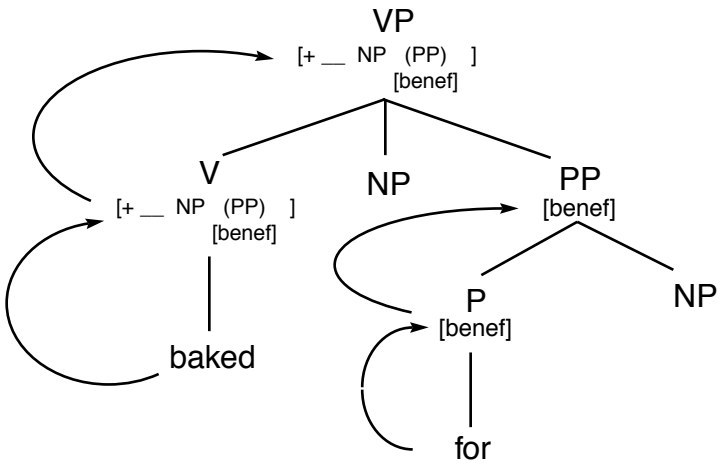
Inheritance of complement information plays a crucial role in enforcing co-occurrence restrictions. This is because of our final principle:

5. The complement features of a lexical node are checked by the mother of that lexical node.

In order for complement information about a lexical item to restrict the trees in which it appears, the information must be passed up the tree as shown above. A useful way to summarize the situation is:

> Syntactica checks for the presence of an item and its complements at the first node in the tree where both the item and its complements are present.

If V is specified as the head of VP, then complement information about *baked* will be passed up, and Syntactica will check for the presence of an NP and an optional benefactive PP under VP. Notice that to satisfy this requirement, the [benef] feature of *for* must also have been passed up correctly:

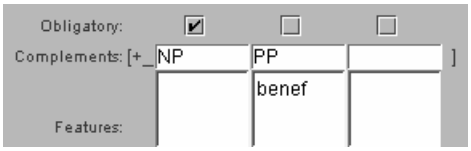


Common Errors in Specifying Complements

The complex interaction of rules, features and complements means that mistakes can easily occur when specifying complements.

Heads Not Specified Correctly for the Complement Configuration

EXAMPLE: Suppose the complements of *baked* are entered as shown below:

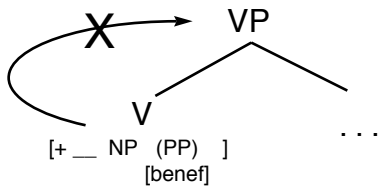


Suppose further that the following rule for VP is entered, where no category is specified as a head:

VP ⟶ V NP PP

Result: Syntactica will incorrectly generate trees containing *baked* and a complement PP that is not [benef].

Reason: V is not specified as a head in the rule, so its complement features are not inherited upward:

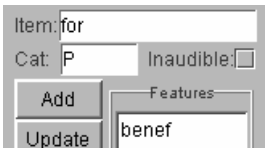


Syntactica treats this situation as equivalent to *baked* placing no constraints on NP and PP in this structure. Anything goes!

Lesson 1: Always specify heads correctly in the rule that generates the complement configuration.

Heads Not Specified Correctly in the Complement Categories

EXAMPLE: Suppose the complements of *baked* are entered as above and suppose *for* is entered as follows:



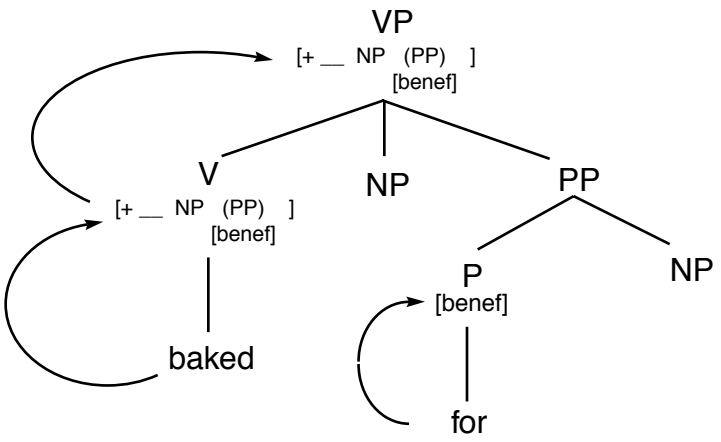
Suppose further that the following rules for VP and PP are given (where boldface indicates a head):

VP → **V** NP PP

PP → P NP

Result: Syntactica will fail to generate trees containing *baked*, an NP and a complement PP containing *for*.

Reason: P is not specified as a head in the PP rule, so its complement features are not passed up to PP. This means that when Syntactica checks for a complement PP with the feature [benef], it does not find one:



Lesson 2: Always specify heads correctly in the rules for the complement categories!

Heads Not Specified Correctly Elsewhere

EXAMPLE: Suppose *for* and *baked* are entered as above, and that the rules for VP and PP are entered correctly with their heads:

VP → **V** NP PP

PP → **P** NP

Suppose further, however, that a rule for VP is also included with no head specified:

VP → V

Result: Syntactica will generate phrase-markers containing *baked* and no complements (as in *John baked*).

Reason: The second VP rule does not declare V as a head, hence any verb occurring in that structure passes no features. Since no features are passed, no complement features are passed. Hence Syntactica enforces no complement restrictions on *baked* in this structure.

Lesson 3: Always specify heads correctly in every rule in which the complement- taking element can occur!

CHAPTER 11

Head-Final Languages

Syntactica will generate trees not only for languages like English, in which the head of a phrase precedes the complements that it selects, but also for languages like Japanese and Hindi, in which heads follow their complements. The former are sometimes called **head-initial languages**, and the latter, **head-final languages**.

When Syntactica is generating trees using rules alone, no changes are required in dealing with head-initial versus head-final languages. Phrase structure rules are entered in the usual way. However, when Syntactica is generating trees using rules and lexicons together, the procedures for head-initial and head-final languages diverge. In particular:

- Rules are entered using the same rule window, but following different conventions.
- Lexical items are created using a different lexicon window, following different conventions for entering information

Entering Head-Final Rules

When Syntactica is using both rules and lexicons to generate trees, rules must be entered in a different way for head-final languages versus head-initial languages.

With a head-initial language, the procedure is as follows:

1. Enter the category of the mother node on the left of the arrow in the rule template.
2. Enter the categories of the daughter nodes on the right of arrow starting with the leftmost field and proceeding rightward with no breaks.

With a head-initial language, however, the procedure is:

1. Enter the category of the mother node on the left of the arrow in the rule template.
2. Enter the categories of the daughter nodes on the right of arrow, starting with the right-most field and proceeding leftward with no breaks.

EXAMPLE: The English VP rule in (1) is entered as shown in the rule template below it:

1. VP → V PP NP

By contrast, the corresponding Japanese VP rule in (2) is entered as shown in the rule template below it:

VP → PP NP V

The specifications for Japanese are essentially the inverse of English.

USER NOTE: These conventions apply only when Syntactica is using both rules and lexicons to generate structures. When rules alone are being used, the conventions for head-final and head-initial languages are the same.

Creating a Head-Final Lexicon

Syntactica handles head-final lexicons differently from head-initial lexicons in two important ways:

- A different lexicon window is accessed
- Complement information is entered differently

The Head-Final Lexicon Window

The default lexicon window launched by Syntactica displays a head-initial complement template. This is the meaning of the notation:

[+ ____ ...]

The lexical item whose complements are to be specified is viewed as appearing in the underline position. Complements then follow the lexical item that they are the complements of.

Syntactica accommodates head-final languages by means of an alternate lexicon window. This alternate window is accessed using the Inspector panel. Open a new lexicon window and choose **Options > Inspector**. The following panel appears:

Change the radio button to Head Final, and click Set (or `press Return`). The lexicon window changes to show the head final configuration of complements:

Note carefully that the complement template now looks like this

[...___ +]

Complements are now understood as preceding the lexical item they are the selected by.

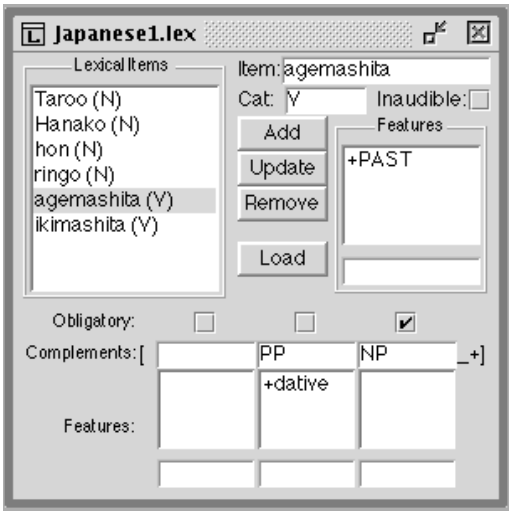
Entering Complements

Complements in a head-final lexicon window are entered in an order opposite to that of a head-initial window. The closest complement to the head is entered in the right most empty field. The next complement is entered in the next empty field to its left. And so on. Thus an item selecting one complement will show only the right most field occupied in a head-final lexicon window. By contrast, in a head-initial lexicon window an item selecting one complement will show only the leftmost field occupied.

EXAMPLE: The lexicon window below shows the entry for English *gave*. English is a head-initial language:

The order of complements reflects the fact that in English, the verb precedes its object (NP), which in turn precedes a dative prepositional phrase (a PP containing *to*).

EXAMPLE: The lexicon window below shows the entry for Japanese *agemashita* 'gave'. Japanese is a head-final language:



The order of complements reflects the fact that in Japanese the verb is preceded by its object (NP), which is in turn preceded by a dative prepositional phrase (a PP containing *ni*).

Head Finality is Global

Syntactica treats head finality as a global property of a lexicon file. It therefore does not allow for the possibility of "mixed lexicons", where some expressions precede their complements and others follow them. Furthermore, once a lexicon file has been specified as head-initial or head-final, and items have been entered into it and saved, head-direction cannot be reset.

Index

A

Active windows 17
  closing 65, 81, 87
  lexicons 17
  list 17, 64
  rules 17
  updating 64

B

Bounding nodes
  and Subjacency 107
  setting 108
Building trees, see Generating phrase markers
Button bar
  including lexical items 53, 72, 85
  including rules 38
  including sentences 80
  including trees 86
  Inspector panel 40, 69
  new lexicon windows 44
  new rule windows 32
  new sentence windows 76
  new tree windows 82
  opening files 12
  position 19
  saving files 16
  show transforms 25
  sorting lexical items 52
  sorting rules 37
  sorting sentences 79

C

Capitalization, in input to TreeViewer 67
Complements
  common errors in specifying 126
  complement template 47
  enforcing co-occurrence restrictions with 125
  entering 47
  entering in head-final languages 132
  inheritance of complement information 123
  interaction with features & rules 123
  multiple 49
  specifying category of 47
  specifying obligatoriness of 47
Constraints on transformations
  with copying
    compared to substitution constraints 119
  with movement
    Identical Source Node Constraint on ATB movement 112
    Subjacency 107
  with substitution
    compared to copying constraints 119
    Empty Target Node Constraint 113
    Like-category Constraint 115
Conventions used in this manual 3

E

Editing windows and files 16
Empty symbol, setting 105
Empty Target Node Constraint 113

F

Features
  agreement in 55
  interaction with rules & complements 123
  interactions with rules 121
  values of 55
Files, see Syntactica files

G

Generating phrase markers 8
  capitalization in input 67
  Error panel 68
  inputting expressions 67
  with rules and a lexicon 24
Grammar as Science Project 2
Grammars
  creating 88
  editing 89
  loading 89
  opening 87
  saving 89
  what are they? 87

H

Head direction, setting 41
Head-final languages
  creating lexicon with 131
  entering lexical items
    complements 132
    lexicon window 131
  entering rules with 129
Heads

declaring 39
setting default direction 41

I

Identical Source Node Constraint on ATB movement 112
Including
lexicons 52
rules 37
sentences 79
trees 85
Installation procedure 5

L

Lexical items
adding to lexical item list 50
compared to dictionary entries 58
complements
entering 47
inheritance of complement information 123
obligatoriness of 47
deleting from lexical item list 50
editing 50
entering
in complement template 47
in lexical item template 46
entering in head-final languages 131
features
values of 55
how they are distinguished 56
including 52
printing 51
saving 50
sorting 52
Lexicon window 43
active 17
for head-final languages 131
Inspector panel
head-initial/head-final setting 131
pleonastic words 113
parts of
complement template 45
lexical item list 44
lexical item template 45
Lexicons
creating with head-final languages 131
editing 50
including 52
loading 63
opening 43
saving 50
sorting 52
Like-category Constraint 115
Loading 16
and closing 65
grammars 89
lexicons 63
rules 63
sentences 79

O

Opening
files 11
grammars 87
lexicon files 43
rule files 19, 31
sentence files 75
tree files 81

P

PF display
declaring items as inaudible 72
in TreeViewer window 63
inherently inaudible items 72
Phrase markers
transforming, see Transforming phrase markers
Pleonastic words
and copying 119
and substitution 113
defined 113
setting 113
Printing
lexicons 51
rules 35
sentences 77
trees 84

R

Root node, setting 40
Rule files
defaults
general 40
head direction 41
root node 40
editing 34
including 37
loading 63
opening 19, 31

printing 35
saving 34
sorting 37
Rule window 31
active 17
Inspector panel
bounding nodes 108
head direction 41
root node 40
variable categories 115
parts of
rule list 32
rule template 32
Rules
adding to rule list 34
changing heads 39
deleting from rule list 34
editing 34
entering 33
entering with head-final languages 129
generating triangles 71
including 37
interaction with features 121
interaction with features & complements 123
loading 63
printing 35
saving 34
setting heads 39
sorting 37

S

Saving 16
grammars 89
lexicons 50
rules 34
sentences 77
Sending
sentences 74, 81
trees 74, 87
Sentence files
including 79
loading 17, 79
opening 75
sorting 79
Sentence window 76
parts of
Load button 76
Send S button 76, 81
sentence field 76
sentence list 76
Sentences
adding to list 77
deleting from list 77
editing 77
including 79
loading 79
printing 77
saving 77
sending 81
sorting 79
Shortcuts
including lexical items 53, 72, 85
including rules 38
including sentences 80
including trees 86
Inspector panel 40, 69
new lexicon windows 44
new rule windows 32
new sentence windows 76
new tree windows 82
opening files 12
saving files 16
show transforms 25
sorting lexical items 52
sorting rules 37
sorting sentences 79
Sorting
lexicons 52
rules 37
sentences 79
Subjacency constraint
defined 107
setting bounding nodes for 108
with adjunctions 111
with substitutions 108
Syntactica
file & window types 11
general layout 10
generating phrase markers 8
how it can be used 2
installation procedure 5
relations between windows 10
start up screen 19
system requirements 5
transforming phrase markers 9
two core functions of 7
what is it? 1
Syntactica files
active windows 17
creating new files 13

editing 16
entering information 14
lexicon 11
loading 16
loading sentence files 17
loading tree files 17
opening 11
rule 11
saving 16
sentence 11
tree 11

T

Trace symbol, setting 105
Transformations
canceling 104
constraints on copying 117
compared to substitution constraints 119
constraints on movement
Identical Source Node Constraint on ATB movement 112
Subjacency 107
constraints on substitution
compared to copying constraints 119
Empty Target Node Constraint 113
Like-category Constraint 115
copying 28, 95
deletion 26, 93
general procedure for performing 92
indexing 93
left-adjunction 97
across-the-board 99
multiple selection with 98
movement operations 92
constraints on 107
Identical Source Node Constraint on ATB movement 112
left-adjunction 97
right-adjunction 100
Subjacency Constraint 107
substitution 101
right-adjunction 100
setting properties of
bounding nodes 108
empty symbol 105
trace symbol 105
variable categories 115
source node 28
substitution 101
across-the-board 103
multiple selection with 102
target node 27, 92
types of
binary 92
unary 92
undoing 29, 104
Transforming phrase markers 9, 25
Transforms panel 25
parts of 91
Status field 104
Tree files
including 85
loading 17
opening 81, 82
saving 84
Tree window 82
parts of
Send T button 87
tree display 82
tree list 82
Trees
adding to list 83
changing format 69
deleting from list 84
displaying triangles 71
including 85
printing 84
saving as EPS images 72
saving as file 84
saving folders of EPS images 84
sending 74, 87
TreeViewer
changing display format 69
generating trees with 65
inputting expressions into 67
loading
lexicons 63
rules 63
sentences 80
trees 86
sending
sentences 74
trees 74
updating loaded windows 64
viewing alternative structures in 22, 68
TreeViewer window 61
TreeViewer window, parts of
active windows list 63
Build and Send buttons 63
PF display 63, 72

sentence field 63
tree display 63
tree list 23, 62
Zoom button 63
Triangles
displaying in trees 71

U

Undoing a transformation 29

V

Variable categories
and copying 119
and substitution 115
defined 115
setting 115

W

Window
Lexicon 43
Sentence 76
Tree 82
TreeViewer 61
Windows
active 17
Lexicon 11
Rule 11, 31
Sentence 11
Tree 11
TreeViewer 11

X

XSB Prolog 6

Z

Zooming
in TreeViewer window 63